Algorithmische Aspekte des Lokalen Lovász Lemmas

Diplomarbeit

Arne Moritz Harff
Goethe-Universität Frankfurt am Main
FB 12 Informatik und Mathematik

8. Mai 2012

für Noreen

Zusammenfassung

Im Rahmen dieser Arbeit wird der aktuelle Stand auf dem Gebiet des Lokalen Lovász Lemmas (*LLL*) beschrieben und ein Überblick über die Arbeiten zu konstruktiven Beweisen und Anwendungen gegeben. Ausgehend von Józef Becks Arbeit zu einer algorithmischen Herangehensweise [2], haben sich in den letzten Jahren im Umfeld von Moser und Tardos und ihren Arbeiten zu einem kostruktiven Beweis des *LLL* eine erneute starke Beschäftigung mit dem Thema und eine Fülle von Verbesserungen entwickelt.

In Kapitel 1 wird als Motivation eine kurze Einführung in die *probabilistische Methode* gegeben. Mit der *First-* und *Second Moment Method* werden zwei einfache Vorgehensweisen vorgestellt, die die Grundidee dieses Beweisprinzips klar werden lassen. Von Paul Erdős eröffnet, beschreibt es Wege, Existenzbeweise in nicht-stochastischen Teilgebieten der Mathematik mithilfe stochastischer Überlegungen zu führen. Das Lokale Lemma als eine solche Überlegung entstammt dieser Idee.
In Kapitel 2 werden verschiedene Formen des *LLL* vorgestellt und bewiesen, außerdem wird anhand einiger Anwendungsbeispiele die Vorgehensweise bei der Verwendung des *LLL* veranschaulicht.
In Kapitel 3 werden algorithmische Herangehensweisen beschrieben, die geeignet sind, von der (mithilfe des *LLL* gezeigten) Existenz gewisser Objekte zur tatsächlichen Konstruktion derselben zu gelangen.
In Kapitel 4 wird anhand von Beispielen aus dem reichen Schatz neuerer Veröffentlichungen gezeigt, welche Bewegung nach der Arbeit von Moser und Tardos entstanden ist. Dabei beleuchtet die Arbeit nicht nur einen anwendungsorientierten Beitrag von Haeupler, Saha und Srinivasan, sondern auch einen Beitrag Terence Taos, der die Beweistechnik Mosers aus einem anderen Blickwinkel beleuchtet.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig abgefasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Ich erkläre ferner, dass diejenigen Stellen der Arbeit, die anderen Werken wörtlich oder dem Sinne nach entnommen sind, in jedem einzelnen Falle unter Angabe der Quellen kenntlich gemacht sind.

Bad Homburg, den 8. Mai 2012

Inhaltsverzeichnis

1 Einleitung - Die probabilistische Methode

Die probabilistische Methode ist ein Lösungsansatz für viele Probleme in Teilgebieten der Mathematik, die nicht unbedingt einen Zusammenhang mit der Stochastik haben und für Fragestellungen und Probleme, die in sich noch keinerlei Zufall enthalten. Dabei gibt es nicht die probabilistische Methode; vielmehr ist damit eine ganze Reihe von Techniken gemeint, die auf dieser Idee beruhen und deren Zahl noch immer wächst.

Die zentrale Idee ist, auf der Suche nach einem bestimmten Konstrukt (beispielsweise einem Graph mit einer gewünschten Eigenschaft) für den Existenzbeweis einen geeigneten Wahrscheinlichkeitsraum zu konstruieren und eine Methode anzugeben, wie die zufällige Erstellung der Konstrukte darin ablaufen kann. Lässt sich zeigen, dass das gesuchte Konstrukt mit positiver Wahrscheinlichkeit auftritt, so ist seine Existenz gezeigt.

1.1 *First Moment Method* - Mit dem Erwartungswert zum Ziel

Eine der einfachsten Techniken der probabilistischen Methode arbeitet mit dem Erwartungswert. Dabei sind abhängig von der Gestalt der Anwendung und der dazu definierten Zufallsvariablen verschiedene Schlussweisen möglich. Ein häufig verwendetes Hilfsmittel ist die Linearität des Erwartungswertes. Lässt sich die Zufallsvariable X als Linearkombination

$$X = c_1 X_1 + \ldots + c_n X_n$$

einer Anzahl von Zufallsvariablen (z.B. Indikatorvariablen) $X_1, \ldots, X_n$ darstellen, so ist

$$\mathbb{E}[X] = c_1 \mathbb{E}[X_1] + \ldots + c_n \mathbb{E}[X_n].$$

Nehmen wir an, dass durch eine von uns definierte Zufallsvariable X die Anzahl gewisser Konstrukte gegeben wird. Lässt sich X in Indikatoren für die Existenz einzelner Konstrukte zerlegen, und kann nun der Erwartungswert aus diesen einzelnen Indikatoren berechnet werden, so ist das Finden einer ersten Schranke (für die Gesamtzahl der beschriebenen Konstrukte) in vielen Problemen einfach. Es gilt: Im Wahrscheinlichkeitsraum existieren Punkte, für die $X \leq \mathbb{E}[X]$ ist und solche mit $X \geq \mathbb{E}[X]$, beziehungsweise

$$\mathbb{E}[X] \leq t \Rightarrow \mathbb{P}(X \leq t) > 0.$$

Für diskrete Zufallsvariablen mit Werten in $\mathbb{N}_0$ (wie zum Beispiel unsere oben beschriebene Zählvariable) bedeutet das insbesondere:

$$\mathbb{E}[X] < 1 \Rightarrow \mathbb{P}(X = 0) > 0.$$

Die folgenden Beispiele zeigen, wie einfach und eingängig diese Beweismethode ist.

1.1.1 Beispiel: Färben von Hypergraphen

Das ursprüngliche *LLL* wurde in [3], einer Arbeit über das Färben von Hypergraphen, veröffentlicht. Es wurde verwendet, um die Existenz einer bestimmten Färbung zu zeigen. Eine Verwendung in diesem Umfeld soll als erstes Beispiel dienen. In Kapitel 2 werden wir das *LLL* vorstellen und zeigen, wie sich das hier dargestellte Ergebnis für gewisse Graphen verbessern lässt.

Ein Hypergraph $G = (E, K)$ ist ein ungerichteter Graph auf der Eckenmenge E. Allerdings verbindet eine Kante aus der Kantenmenge K nicht unbedingt nur zwei Ecken, sondern kann mehrere Ecken umfassen. Mit $|k|$ bezeichnen wir die Zahl der durch k verbundenen Ecken.
Wir färben die Ecken des Hypergraphen mit m Farben. Eine Kante $k \in K$ heißt *monochrom*, wenn alle Ecken aus k dieselbe Farbe erhalten, eine Färbung heißt *gültig*, falls keine der Kanten monochrom ist. Ein Graph, für den eine gültige Färbung mit m Farben existiert, heißt *m-färbbar*.

Behauptung 1.1. *Sei $G = (E, K)$ ein Hypergraph mit $|K| < 2^{n-1}$ und sei $n = \min_{k \in K} |k|$. Dann existiert eine gültige 2-Färbung für G.*

Beweis. Wir färben alle Ecken unabhängig voneinander jeweils mit $p = 1/2$ rot und mit $q = 1/2$ blau. Für jede Kante $k \in K$ definieren wir die Indikatorvariable I_k mit

$$I_k = \begin{cases} 1, & \text{falls } k \text{ monochrom.} \\ 0, & \text{falls } k \text{ nicht monochrom.} \end{cases}$$

Weiterhin definieren wir die Zufallsvariable $X = \sum_{k \in K} I_k$, die die Anzahl der monochromen Kanten in G zählt. Für alle I_k gilt: $\mathbb{E}[I_k] = \mathbb{P}(I_k \text{ monochrom})$. Und damit

$$\mathbb{E}[I_k] = 2^{-(|k|-1)} \leq 2^{-(n-1)}.$$

Das bedeutet für $\mathbb{E}[X]$:

$$\begin{aligned} \mathbb{E}[X] = \sum_{k \in K} \mathbb{E}[I_k] &\leq \sum_{k \in K} 2^{-(n-1)} \\ &= |K| \cdot 2^{-(n-1)} \\ &< 2^{n-1} \cdot 2^{-(n-1)} = 1. \end{aligned}$$

Da X ausschließlich Werte aus $\mathbb{N}_0$ annimmt, können wir schließen:

$$\mathbb{E}[X] < 1 \Rightarrow \mathbb{P}(X = 0) > 0.$$

Durch zufälliges Färben der Ecken erreichen wir mit positiver Wahrscheinlichkeit eine Färbung ohne monochrome Kanten, d.h. eine gültige Färbung von G. □

1.2 *Second Moment Method* - Varianz und Chebyshev-Ungleichung

Die *Second Moment Method* als weitere Ausprägung der probabilistischen Methode arbeitet mit der Varianz σ^2, die entscheidend durch das zweite Moment $\mathbb{E}[X^2]$ einer Variable beeinflusst wird ($\sigma^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2$).
Mit der *Chebyshev*-Ungleichung kennen wir eine einfache Konzentrationsungleichung, die die Varianz einer Zufallsvariablen verwendet. Die Nutzung dieser Ungleichung bezeichnen wir in unserem Zusammenhang als *Second Moment Method*. Für die Anwendung interessant wird diese, wenn die oben vorgestellte *First Moment Method* nicht verwendet werden kann; etwa, wenn gezeigt werden soll, dass X mit hoher Wahrscheinlichkeit positiv ist oder dass diese Wahrscheinlichkeit für $n \to \infty$ gegen 0 geht.

Lemma 1.2 (Chebyshev-Ungleichung)**.** *Für alle $t > 0$ gilt:*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq t\sigma) \leq \frac{1}{t^2}.$$

Mit dem Lemma folgt für $t = \dfrac{\mathbb{E}[X]}{\sigma}$, dass

$$\mathbb{P}(X = 0) \leq \frac{\sigma^2}{\mathbb{E}[X]^2}$$

und damit ergibt sich direkt die folgende

Beobachtung 1.3. *Wenn für eine Folge Y_n nichtnegativer Zufallsvariablen $\sigma^2 = o(\mathbb{E}[Y_n]^2)$ gilt, dann ist mit hoher Wahrscheinlichkeit $Y_n > 0$. Die Voraussetzung lässt sich äquivalent als*

$$\frac{\mathbb{E}[Y_n^2]}{\mathbb{E}[Y_n]^2} \longrightarrow 1 \quad \textit{für } n \to \infty$$

formulieren. Für eine Folge von Zufallsvariablen definieren wir „A tritt mit hoher Wahrscheinlichkeit ein.“ für ein Ereignis A als $\mathbb{P}(A) \to 1$ für $n \to \infty$.

1.2.1 Beispiel: Isolierte Ecken in zufälligen Graphen

Für dieses Beispiel müssen zunächst wieder einige Begriffe kurz erklärt werden. Ausführlichere Beschreibungen zur Idee des zufälligen Graphen finden sich unter anderem bei Alon und Spencer in [1].
Der *zufällige Graph* $G(n,p)$ ist eine Zufallsvariable in die Menge der Graphen mit n Ecken. Der Weg zu einem zufällig aus dieser Menge gegriffenen Graphen ist einfach: Jede der $\binom{n}{2}$ möglichen Kanten ist unabhängig von den anderen mit Wahrscheinlichkeit p in G vorhanden und mit Wahrscheinlichkeit $q = (1-p)$ nicht.

Eine Ecke e_i in der Eckenmenge E eines Graphen G heißt *isoliert*, wenn im Graphen keine Kante $\{e_i, e_j\}$ existiert, die e_i mit einer anderen Ecke $e_j \in E$ verbindet.

Behauptung 1.4. *Im zufälligen Graphen $G(n,p)$ mit Eckenmenge E und $p = \frac{c \cdot \ln n}{n}$ gilt für die Anzahl Y_n der isolierten Ecken:*

$$\mathbb{P}(Y_n > 0) \underset{n\to\infty}{\longrightarrow} \begin{cases} 0 & \text{für } c > 1, \\ 1 & \text{für } c < 1. \end{cases}$$

Beweis. Wir berechnen zunächst den Erwartungswert von Y_n als Summe der n Indikatorvariablen X_i (mit $X_i = 1$, falls die Ecke e_i isoliert ist). Dabei ist $\mathbb{E}[X_i] = \mathbb{P}(X_i = 1) = (1-p)^{(n-1)}$. Diese Wahrscheinlichkeit ergibt sich, da alle $(n-1)$ möglichen Kanten mit Beteiligung von e_i nicht im zufälligen Graph vorkommen dürfen. Wir erhalten für $\mathbb{E}[Y_n]$:

$$\begin{aligned} \mathbb{E}[Y_n] &= \mathbb{E}\left[\sum_{i\in E} X_i\right] \\ &= \sum_{i\in E} \mathbb{E}[X_i] \\ &= n\,(1-p)^{(n-1)}. \end{aligned}$$

Es folgt für $n \to \infty$:

$$\mathbb{E}[Y_n] = n\left(1 - \frac{c\cdot\ln n}{n}\right)^{(n-1)} \sim ne^{-(c\cdot\ln n)/n} = n^{(1-c)},$$

sodass wir entsprechend der Werte von c unterscheiden können:

$$\mathbb{E}[Y_n] \underset{n\to\infty}{\longrightarrow} \begin{cases} 0 & \text{für } c > 1, \\ \infty & \text{für } c < 1. \end{cases}$$

Damit haben wir für $c > 1$ bereits die Behauptung gezeigt. Da $\mathbb{E}[Y_n] \to 0$, geht auch die Wahrscheinlichkeit $\mathbb{P}(Y_n > 0)$ gegen 0.

Für $c < 1$ liegt der Fall nicht so einfach. Zwar geht der Erwartungswert gegen ∞, es kann allerdings trotzdem noch nichts über $\mathbb{P}(Y_n > 0)$ gesagt werden. Wir berechnen nun zur Verwendung unserer Beobachtung 1.3 das zweite Moment $\mathbb{E}[Y_n^2]$.

$$\begin{aligned} Y_n^2 &= \left(\sum_{i \in E} X_i\right)^2 \\ &= \sum_{(i,j) \in E} X_i X_j \\ &= \sum_{i \in E} X_i^2 + \sum_{(i,j) \in E, i \neq j} X_i X_j \\ &= \sum_{i \in E} X_i + \sum_{(i,j) \in E, i \neq j} X_i X_j. \end{aligned}$$

So ergibt sich für den Erwartungswert

$$\begin{aligned} \mathbb{E}\left[Y_n^2\right] &= \sum_{i \in E} \mathbb{E}[X_i] + \sum_{(i,j) \in E, i \neq j} \mathbb{E}[X_i X_j] \\ &= n\,(1-p)^{(n-1)} + n(n-1)\,(1-p)^{(2(n-1)-1)} \\ &= n\,(1-p)^{(n-1)} \left(1 + (n-1)\,(1-p)^{(n-2)}\right). \end{aligned}$$

Wir prüfen nun die Voraussetzung aus Beobachtung 1.3

$$\begin{aligned} \frac{\mathbb{E}[Y_n^2]}{\mathbb{E}[Y_n]^2} &= \frac{n\,(1-p)^{(n-1)} \left(1 + (n-1)\,(1-p)^{(n-2)}\right)}{\left(n\,(1-p)^{(n-1)}\right)^2} \\ &= \frac{1 + (n-1)\,(1-p)^{(n-2)}}{n\,(1-p)^{(n-1)}} \\ &= \underbrace{\frac{1}{n\,(1-p)^{(n-1)}}}_{\longrightarrow 0} + \underbrace{\frac{n-1}{n\,(1-p)}}_{\longrightarrow 1} \qquad \underset{n \to \infty}{\longrightarrow} 1. \end{aligned}$$

Somit haben wir den zweiten Teil der Behauptung mithilfe der *Second Moment Method* bewiesen. □

2 Das Lokale Lovász Lemma

Das *Lokale Lovász Lemma (LLL)* wurde 1975 von László Lovász und Paul Erdős in [3] vorgestellt. In seiner Urform lautet es wie folgt:

Lemma 2.1. *Sei G ein (endlicher) Graph mit maximalem Grad d und Ecken $v_1, \ldots, v_n$. Wir wollen ein Ereignis A_i mit v_i $(i = 1, \ldots, n)$ verknüpfen und nehmen an, dass A_i unabhängig ist von der Menge*

$$\{A_j : (v_i, v_j) \notin E(G)\}.$$

Wir nehmen weiter an, dass

$$\mathbb{P}(A_i) \leq \frac{1}{4d}.$$

Dann gilt:

$$\mathbb{P}(\bar{A}_1 \ldots \bar{A}_n) > 0.$$

2.1 Formen des Lokalen Lovász Lemmas

Satz 2.2 (Das allgemeine Lokale Lovász Lemma)**.** *Sei $\mathcal{A}$ eine endliche Familie von Ereignissen. Für $A \in \mathcal{A}$ sei $\Gamma(A)$ eine Teilmenge von $\mathcal{A}$, sodass A unabhängig von $\mathcal{A} \setminus (\Gamma(A) \cup \{A\})$ ist.*
Existiert eine Zuordnung $x : \mathcal{A} \to (0,1)$ $(x_A := x(A))$ mit

$$\mathbb{P}(A) \leq x_A \prod_{B \in \Gamma(A)} (1 - x_B) \quad \forall A \in \mathcal{A},$$

so ist

$$\mathbb{P}\left(\bigcap_{A \in \mathcal{A}} A^c\right) \geq \prod_{A \in \mathcal{A}} (1 - x_A),$$

und damit insbesondere echt positiv.

Beweis. Wir betrachten eine Menge $S \subset \mathcal{A}$ von Ereignissen. Durch Induktion über $s = |S|$ zeigen wir für alle S:

$$\mathbb{P}(A \mid \bigcap_{B \in S} B^c) \leq x_A \quad \forall A \in \mathcal{A}. \tag{1}$$

O.B.d.A. nummerieren wir die Ereignisse $A \in \mathcal{A}$ als $\mathcal{A} = \{A_1, \ldots, A_n\}$, wobei $A = A_n$, $\Gamma(A_n) = \{A_1, \ldots, A_d\}$ und $S = \{A_1, \ldots, A_s\}$.
Für $s = 0$ ist für alle $A \in \mathcal{A}$

$$\mathbb{P}(A \mid \bigcap_{B \in S} B^c) \;=\; \mathbb{P}(A) \;\leq\; x_A \prod_{B \in \Gamma(A)} (1 - x_B) \;\leq\; x_A.$$

Für $s = 1$ betrachten wir zwei Fälle:

i) Für $S \nsubseteq \Gamma(A_n)$ gilt trivialerweise

$$\mathbb{P}(A_n \mid A_1^c) = \mathbb{P}(A_n) \leq x_A.$$

ii) Für $S \subset \Gamma(A_n)$ gilt:

$$\mathbb{P}(A_n \mid A_1^c) = \frac{\mathbb{P}(A_n \cap A_1^c)}{\mathbb{P}(A_1^c)} .$$

Der Zähler kann nun durch $\mathbb{P}(A_n \cap A_1^c) \leq \mathbb{P}(A_n)$ abgeschätzt werden, der Nenner durch

$$\begin{aligned} \mathbb{P}(A_1^c) &\geq 1 - x_{A_1} \prod_{B \in \Gamma(A_1)} (1 - x_B) \\ &\geq 1 - x_{A_1} \\ &\geq \prod_{B \in \Gamma(A_n)} (1 - x_B). \end{aligned}$$

Und damit ist nach Voraussetzung $\mathbb{P}(A_n \cap A_1^c) \leq x_{A_n}$.

Nehmen wir nun an, Gleichung (1) sei für alle S mit $|S| < s$ und $s \geq 1$ erfüllt. Um die Induktion zu beenden, betrachten wir also den Fall $|S| = s$:

$$\mathbb{P}(A_n \mid A_1^c, \ldots, A_s^c) = \frac{\mathbb{P}(A_n \cap A_1^c \cap \ldots \cap A_d^c \mid A_{d+1}^c, \ldots, A_s^c)}{\mathbb{P}(A_1^c \cap \ldots \cap A_d^c \mid A_{d+1}^c, \ldots, A_s^c)} .$$

Der Zähler wird abgeschätzt durch

$$\begin{aligned} \mathbb{P}(A_n \cap A_1^c \cap \ldots \cap A_d^c \mid A_{d+1}^c, \ldots, A_s^c) &\leq \mathbb{P}(A_n \mid A_{d+1}^c, \ldots, A_s^c) \\ &= \mathbb{P}(A_n), \end{aligned}$$

der Nenner durch

$$\begin{aligned} \mathbb{P}(A_1^c \cap \ldots \cap A_d^c \mid A_{d+1}^c, \ldots, A_s^c) &= \prod_{i=1}^{d} \mathbb{P}(A_i^c \mid A_{i+1}^c, \ldots, A_s^c) \\ &\overset{(*)}{\geq} \prod_{i=1}^{d} (1 - x_{A_i}) = \prod_{B \in \Gamma(A_n)} (1 - x_B), \end{aligned}$$

wobei für $(*)$ die Induktionsannahme verwendet wird:

$$\mathbb{P}(A_i^c \mid A_{i+1}^c, \ldots, A_s^c) = 1 - \mathbb{P}(A_i \mid A_{i+1}^c, \ldots, A_s^c) \geq 1 - x_{A_i}.$$

Beide Abschätzungen liefern uns in Kombination den Induktionsschluss

$$\mathbb{P}(A_n \mid A_1^c, \ldots, A_s^c) \;\leq\; \frac{\mathbb{P}(A_n)}{\prod_{B\in\Gamma(A_n)}(1-x_B)} \;\leq\; x_{A_n}.$$

Es folgt mithilfe der Abschätzung aus Gleichung (1):

$$\begin{aligned}
\mathbb{P}\left(\bigcap_{A\in\mathcal{A}} A^c\right) &= \prod_{i=1}^{n} \mathbb{P}(A_i^c \mid A_1^c, \ldots A_{i-1}^c) \\
&= \prod_{i=1}^{n} (1 - \mathbb{P}(A_i \mid A_1^c, \ldots A_{i-1}^c)) \\
&\geq \prod_{i=1}^{n} (1 - x_{A_i}) \\
&= \prod_{A\in\mathcal{A}} (1 - x_A) \quad > 0.
\end{aligned}$$

□

Im einfacheren symmetrischen Fall, können wir die folgende Form des *LLL* verwenden, die bis auf die Verbesserung der Konstante der ursprünglichen Form gleicht.

Satz 2.3 (Das symmetrische Lokale Lovász Lemma)**.** *Für eine Menge $\mathcal{A}$ von Ereignissen und $\Gamma(A)$, sodass A unabhängig von $\mathcal{A} \setminus (\Gamma(A)) \cup \{A\}$, gelte für alle $A \in \mathcal{A}$:*

i) $\mathbb{P}(A) \leq p < 1$,

ii) $|\Gamma(A)| \leq d$.

Ist $e \cdot p \cdot (d+1) \leq 1$, so gilt:

$$\mathbb{P}\left(\bigcap_{A\in\mathcal{A}} A^c\right) > 0.$$

Beweis. Das symmetrische *LLL* folgt direkt aus der allgemeinen Form. Da die Aussage für $d = 0$ (also alle A unabhängig) trivial ist, genügt es, $d \geq 1$ zu betrachten. Wir erhalten

$$\begin{aligned}
e \cdot p \cdot (d+1) &\leq 1 \\
\Leftrightarrow p &\leq \frac{1}{(d+1)e} \\
&\leq \frac{1}{d+1}\left(1 - \frac{1}{d+1}\right)^d
\end{aligned}$$

und setzen $x_A = x := \frac{1}{d+1}$ für alle $A \in \mathcal{A}$, sodass (ebenfalls für alle A) gilt:

$$\mathbb{P}(A) \leq \frac{1}{d+1}\left(1 - \frac{1}{d+1}\right)^d \leq x \prod_{B \in \Gamma(A)} (1-x).$$

□

Bemerkung 2.4. *Für das symmetrische* LLL *ist e die bestmögliche Konstante. In vielen Fällen genügt auch die etwas eingängigere Form* $4 \cdot p \cdot d \leq 1$ *aus dem ursprünglichen* LLL.

In einigen Quellen finden sich Formen des *LLL* , die die Abhängigkeiten zwischen den Ereignissen $A \in \mathcal{A}$ mithilfe eines Abhängigkeitsgraphen $G(\mathcal{A}, K)$ beschreiben, wobei eine Kante $k = (A, B) \in K$ genau dann existiert, wenn A und B nicht unabhängig sind. Eine eingehendere Beschreibung dieser Idee findet sich in Kapitel 3.

2.2 Anwendungsbeispiele

2.2.1 k-SAT

Eine *Boolsche Variable* b ist eine Variable, die nur die Werte *wahr* oder *falsch* annehmen kann. Ein *Literal* ist von der Form b oder $\bar{b}$, wobei $\bar{b}$ die Verneinung von b ist; also „nicht b“. Eine Verknüpfung aus mehreren Literalen, die durch $\wedge$ („und“) oder $\vee$ („oder“) verbunden werden, heißt *Klausel*. Untereinander können Klauseln wieder mit $\wedge$ oder $\vee$ verknüpft werden. Eine solche Verknüpfung mehrerer Klauseln heißt *Boolsche Formel*.
Eine Boolsche Formel steht in *konjunktiver Normalform* (CNF), wenn innerhalb der Klauseln alle Literale mit $\vee$ und die Klauseln untereinander mit $\wedge$ verbunden sind. Jede Boolsche Formel kann in CNF gebracht werden.

k-SAT ist das vielfach zitierte Problem, eine Boolsche Formel in konjunktiver Normalform (CNF) mit genau k Literalen pro Klausel auf ihre Erfüllbarkeit zu prüfen. Erfüllbar ist eine Formel dann, wenn eine *wahr/falsch*-Zuordnung zu den Boolschen Variablen existiert, sodass die gesamte Formel *wahr* ist. Mithilfe des *LLL* lassen sich Schranken für die Erfüllbarkeit beweisen.

Behauptung 2.5. *Jede Eingabe von k-SAT, bei der jede Variable in höchstens* $(2^{k-2})/k$ *Klauseln auftritt, ist erfüllbar.*

Beweis. Jede Boolsche Variable der Eingabe von k-SAT erhalte unabhängig von den anderen jeweils mit Wahrscheinlichkeit $1/2$ die Werte „wahr“ oder „falsch“. Für jede Klausel i sei A_i das Ereignis „i ist falsch“, $B(i)$ sei die

Menge der Boolschen Variablen, die in i auftreten. Dann ist $\Gamma(A_i)$ die Menge der Klauseln $j \neq i$ mit $B(i) \cap B(j) \neq \emptyset$, $\mathcal{A} := \{A_i \mid i \text{ Klausel in der Eingabe}\}$. Da jede Variable in höchstens $(2^{k-2})/k$ Klauseln auftritt, ist $|\Gamma(A_i)| \leq 2^{k-2}$, $\mathbb{P}(A_i) = 2^{-k}$ für alle Klauseln i, also kann $x : \mathcal{A} \mapsto (0,1)$ konstant gewählt werden. Es gilt:

$$\begin{aligned} \mathbb{P}(A_i) = 2^{-k} &\leq x_{A_i} \prod_{A_j \in \Gamma(A_i)} \left(1 - x_{A_j}\right) \\ &= x \prod_{A_j \in \Gamma(A_i)} (1-x) \\ &= x\,(1-x)^{|\Gamma(A_i)|}. \end{aligned}$$

Um x zu finden, betrachten wir $x\,(1-x)^{\left(2^{k-2}\right)} \leq \min_{A_i} x\,(1-x)^{|\Gamma(A_i)|}$. Wir suchen x, sodass:

$$2^{-k} \leq x\,(1-x)^{\left(2^{k-2}\right)}.$$

Das ist zum Beispiel für $x = 1/(2^{k-2}+1)$ erfüllt. Damit sind die Voraussetzungen des *LLL* erfüllt und es gilt:

$$\mathbb{P}\left(\bigcap_{A_i \in \mathcal{A}} A_i^c\right) > \prod_{A_i \in \mathcal{A}} (1 - x_{A_i}) > 0.$$

Das heißt: Mit positiver Wahrscheinlichkeit tritt keines der A_i ein und damit ist unsere Eingabe von k-SAT erfüllbar, denn sie ist mit positiver Wahrscheinlichkeit erfüllt. □

2.2.2 Noch einmal: Färben von Hypergraphen

Im ersten Beispiel in Abschnitt 1.1.1 hatten wir mithilfe der *First Moment Method* die 2-Färbbarkeit von Hypergraphen mit nicht zu vielen, aber hinreichend großen Kanten bewiesen. Mit dem *LLL* können wir die Beschränkung für die Anzahl der Kanten fallen lassen und beliebig große Graphen färben, falls die Zahl der Überschneidungen beschränkt werden kann.

Zur Erinnerung: Eine Kante $k \in K$ heißt *monochrom*, wenn alle Ecken aus k dieselbe Farbe erhalten, eine Färbung heißt *gültig*, falls keine der Kanten monochrom ist.

Behauptung 2.6. *Sei $G = (E, K)$ ein Hypergraph und sei $n = \min_{k \in K} |k|$. Wenn eine Kante k höchstens $(2^{n-1}/e) - 1$ andere Kanten schneidet (also gemeinsame Ecken mit ihnen hat), existiert eine gültige 2-Färbung für G.*

Beweis. Für jede Kante $k \in K$ sei A_k das Ereignis „k ist monochrom“. Eine gültige 2-Färbung existiert genau dann, wenn $\mathbb{P}\left(\bigcap_{k\in K} A_k^c\right) > 0$.
Da jede Kante mindestens n Knoten enthält, ist $\mathbb{P}(A_k) \leq 2^{1-n}$ für alle $k \in K$.
Durch die beschränkte Anzahl anderer Kanten, die k schneidet, erhalten wir $|\Gamma(A_k)| \leq (2^{n-1}/e) - 1$.
Wir wenden das symmetrische LLL mit $p = 2^{1-n}$ und $d = (2^{n-1}/e) - 1$ an.
Es gilt offensichtlich

$$e \cdot p \cdot (d+1) = e \cdot 2^{1-n} \cdot \frac{2^{n-1}}{e} \leq 1,$$

damit sind die Bedingungen des symmetrischen LLL erfüllt und die geforderte 2-Färbung von G existiert. □

Bemerkung 2.7. *Die obige Behauptung lässt sich direkt auf das in Beispiel 2.2.1 beschriebene k-SAT-Problem übertragen. Wir identifizieren die Kanten des Hypergraphen mit den Klauseln in der Boolschen Formel, während die Ecken den Boolschen Variablen entsprechen und die Farben der Ecken den Wahrheitswerten der Variablen. Wir erhalten:*
Wenn jede k-Klausel mit höchstens $(2^k/e) - 1$ *anderen Klauseln gemeinsame Variablen hat, ist die Formel erfüllbar.*
Der veränderte Exponent entsteht, da im Hypergraphen zwei monochrome Färbungen vermieden werden müssen, in der Boolschen Formel aber nur eine verbotene Belegung.

2.2.3 Daten in Netzwerken - *packet routing*

Das hier vorgestellte Beispiel lässt sich auf mehrere praktische Anwendungen übertragen. Die grundsätzliche Idee ist das Versenden von Datenpaketen in einem Netzwerk, es lässt sich aber auch auf Anwendungen im Logistikbereich übertragen.
Soll eine Menge von Daten durch ein Netzwerk (einen gerichteten Graphen) geschickt werden, wobei zu jedem einzelnen Datenpaket ein Start- und ein Zielknoten gegeben sind, dann ist das Ziel, diesen Transport möglichst schnell und mit möglichst kleinen Warteschlangen an den einzelnen Knoten auszuführen. Die *Warteschlangen* befinden sich jeweils am Ende der Kanten. Dort werden die Pakete gespeichert, bis sie zum nächsten Knoten gesendet werden. Ist ein Paket an seinem Ziel angekommen, so wird es nicht mehr der Warteschlange hinzugefügt. Ein *Paket* ist dabei eine zusammenhängende Menge von Daten, die nur gebündelt weitergeschickt werden kann. Die Ausführung folgt einem festen Takt, wobei die Pakete pro Schritt entweder an einem Knoten warten oder entlang einer Kante zum nächsten Knoten

transportiert werden können.
Zwei Einschränkungen erschweren diesen Prozess: *(i)* Eine Kante kann in einem Schritt maximal ein Paket transportieren. *(ii)* Eine Kante transportiert ein Paket nur, wenn vor dem Schritt in der Warteschlange am Ende der Kante noch ein Platz frei ist.
Bei der Planung eines solchen Versandprozesses sind zwei Dinge zu erledigen: Es müssen die Wege festgelegt werden, auf denen die Pakete durch das Netzwerk laufen und es muss ein *Zeitplan* erstellt werden, der angibt, welche Pakete wann welche Kanten nehmen. Für dieses Beispiel nehmen wir an, dass die Wege bereits vorgegeben sind und nur noch der Zeitplan erstellt werden muss.
Wichtige Kenngrößen zur Bestimmung des benötigten Zeitaufwands sind die maximale Weglänge d (*dilation*/Ausdehnung) und die maximale Zahl von Paketen, die dieselbe Kante nehmen, c (*congestion*/Stauung). Ein trivialer Zeitplan benötigt also $c \cdot d$ Schritte und Warteschlangen der Größe c an jeder Kante. Tatsächlich geht es viel schneller. Leighton, Maggs und Rao haben 1994 in [6] mithilfe des *LLL* gezeigt, dass für beliebige Netzwerke und beliebige Anzahl von Paketen ein Zeitplan der Länge $O(c + d)$ erstellt werden kann. Wir betrachten hier nur das für den Beweis der Aussage entscheidende Lemma, in dessen Beweis das *LLL* verwendet wird. Für das Lemma wird die Bedingung, nur ein Paket pro Kante und Zeitschritt transportieren zu können, fallen gelassen. Diese Abweichung von den ursprünglichen Bedingungen wird im Beweis der Hauptaussage wieder korrigiert.
Im Lemma werden zwei Begriffe verwendet, die bisher noch nicht definiert wurden. Ein *Zeitrahmen der Länge* L ist eine Abfolge von L aufeinanderfolgenden Zeitschritten. Die *Rahmenstauung* R ist die maximale Anzahl von Paketen, die innerhalb des Zeitrahmens entlang irgendeiner Kante zu transportieren sind. Die *relative Stauung* r in einem Zeitrahmen der Länge L ist das Verhältnis R/L zwischen der Rahmenstauung und der Länge des Zeitrahmens.

Lemma 2.8. *Für jede Menge von Paketen, deren Wege mit Ausdehnung d und Stauung c jede Kante nur einmal pro Paket verwenden, existiert ein Zeitplan der Länge $O(c+d)$, in dem kein Paket unterwegs wartet und der in jedem mindestens $\log d$ langen Zeitrahmen eine relative Stauung von höchstens 1 aufweist.*

Beweis. Für den Beweis nehmen wir o.B.d.A. an, dass $c = d$ ist. Wir weisen jedem Paket unabhängig von den anderen eine zufällige Wartezeit W zu (uniform aus $[1, \alpha d]$). Die Konstante α wird später genauer bestimmt. Ein Paket wartet genau W Schritte an seinem Startknoten und läuft danach ohne unterwegs noch einmal anzuhalten bis zu seinem Zielknoten. Die Länge des

so aufgestellten Zeitplans ist höchstens $(\alpha+1)d$ (nämlich wenn ein Paket mit der maximalen Weglänge d die längstmögliche Wartezeit $W = \alpha d$ zugewiesen bekommt).
Für jede Kante g betrachten wir das zu vermeidende Ereignis A_g, dass innerhalb irgendeines Zeitrahmens der Länge $L \geq \log d$ mehr als L Pakete die Kante g benutzen müssen. A_g hängt ausschließlich von den Wartezeiten ab, die diejenigen Pakete zugewiesen bekommen, die g irgendwann auf ihrem Weg benutzen müssen. Folglich sind zwei Ereignisse A_g und A_h genau dann unabhängig, wenn es kein Paket gibt, dessen Weg sowohl über g als auch über h führt. Eine obere Schranke für die Zahl der Nachbarn von A_g im Abhängigkeitsgraphen können wir also leicht berechnen: g wird von höchstens c Paketen benutzt, jedes dieser Pakete durchläuft höchstens $d-1$ andere Kanten, und mit $c = d$ folgt $|\Gamma(A_g)| \leq d^2$.
Die Wahrscheinlichkeit $\mathbb{P}(A_g)$ eines solchen Ereignisses können wir nach oben abschätzen. Wir summieren dafür die Wahrscheinlichkeit, dass ein Zeitrahmen der Länge L relative Stauung größer 1 hat, über alle relevanten Längen $L \in [\log d, d]$. ($L > d$ muss nicht betrachtet werden, da die Stauung $c = d$ ist und für längere Zeitrahmen eine relative Stauung $r > 1$ nicht erreicht werden kann.)
Die Anzahl der Pakete, die innerhalb des Zeitrahmens der Länge L die Kante g benutzen, ist binomialverteilt. Für jedes der maximal d Pakete, deren Weg über g führt, findet bei der Zuweisung seiner Wartezeit ein Bernoulli-Versuch mit Erfolgswahrscheinlichkeit $L/(\alpha d)$ statt, da nur L der insgesamt αd möglichen Wartezeiten dazu führen, dass dieses Paket die Kante g innerhalb des L-Zeitrahmens benutzt. Die Wahrscheinlichkeit für mehr als L Erfolge lässt sich nach oben abschätzen durch

$$\binom{d}{L}\left(\frac{T}{\alpha d}\right)^T.$$

Insgesamt ergibt sich also für die Wahrscheinlichkeit eines Ereignisses A_g folgende Abschätzung:

$$\mathbb{P}(A_g) \leq p := \sum_{L=\log d}^{d} (1+\alpha)d\binom{d}{L}\left(\frac{T}{\alpha d}\right)^T.$$

Mit p und $|\Gamma(A_g)| \leq d^2$ können wir α einen hinreichend großen festen Wert zuweisen, sodass gilt:

$$4 \cdot p \cdot |\Gamma(A_g)| < 1.$$

Mit dem *LLL* folgt entsprechend Bemerkung 2.4, dass eine Zuordnung von Wartezeiten W zu den Paketen existiert, mit der in jedem Zeitrahmen einer Länge größer $\log d$ die relative Stauung höchstens 1 ist. □

3 Algorithmen

Das *LLL* bietet also, wie wir in Kapitel 2 gesehen haben, die Möglichkeit, die Existenz bestimmter Objekte zu beweisen. Leider erhalten wir aus dem *LLL* aber keine Hinweise darauf, wie diese Objekte zu finden sind. Nachdem Józef Beck 1991 in [2] am Beispiel der 2-Färbungen von Hypergraphen erste algorithmische Ansätze aufgezeigt hatte (wobei er allerdings Einschränkungen in den Bedingungen hinnehmen musste), haben Robin Moser und Gábor Tardos 2010 in [8] für eine geringfügig modifizierte Situation nahezu allgemein anwendbare Algorithmen beschrieben.

3.1 Beck - Der algorithmische Ansatz

Józef Beck stellte in [2] einen algorithmischen Ansatz für das Erstellen einer 2-Färbung von Hypergraphen vor. Die Aufgabe, für Teilmengen H_i einer endlichen Menge X jeweils eine sogenannte „verbotene Färbung“ f_i zu vermeiden, ist dabei allgemeiner als das üblicherweise betrachtete Vermeiden einer monochromen Färbung der Kanten im Hypergraphen. Es können auch mehrere Färbungen pro Teilmenge vermieden werden, indem mit mehreren identischen H_i verschiedene verbotene Färbungen verknüpft werden. An anderer Stelle müssen allerdings die Voraussetzungen des *LLL* abgeschwächt werden: Die in Abschnitt 2.2.2 gezeigte exponentielle Schranke $(2^{n-1}/e) - 1$ für die Anzahl der Überschneidungen mit anderen Kanten wird durch die schwächere Schranke $2^{(n/48)+1}$ ersetzt.
Becks deterministischer sequentieller Algorithmus liefert unter diesen Voraussetzungen in Polynomialzeit eine entsprechende Färbung.

Behauptung 3.1. *Sei X eine endliche Menge und sei $\mathcal{H} = \{H_1, \ldots, H_M\}$ eine Familie nicht notwendigerweise verschiedener n-elementiger Teilmengen von X ($n \geq 2$). Zu jedem $H_i \in \mathcal{H}$ sei eine 2-Färbung $f_i : H_i \to \{$rot, blau$\}$ gegeben. Wenn jedes H_i höchstens $2^{(n/48)+1}$ andere $H_j \in \mathcal{H}$ schneidet, dann findet Becks Algorithmus in Laufzeit M^{const} eine Färbung $f : X \to \{$rot, blau$\}$, sodass für alle $i \in \{1, \ldots, M\}$ gilt:*

$$f|_{H_i} \neq f_i.$$

Wir wollen die Grundzüge des Algorithmus betrachten. Der Ablauf lässt sich folgendermaßen zusammenfassen: Zunächst werden sogenannte *(2, 6)-Bäume* der Ordnung $\frac{\beta \log M}{n}$ gesucht: $\frac{\beta \log M}{n}$-elementige Mengen $\{H_i : i \in J\}$, die sich so anordnen lassen, dass im Abhängigkeitsgraphen (Def. 3.5) der Abstand von $H_{i_{j+1}}$ zur Menge $\{H_{i_1}, \ldots, H_{i_j}\}$ für alle j mindestens 2, höchstens 6 beträgt. Zu jedem solchen Baum werden die Vereinigungsmenge $B(= B_J) = \bigcup_{i \in J} H_i$

und die Färbung $g_B = \bigcup_{i \in J} f_i$ betrachtet. Die Familie dieser B heißt $\mathcal{B}$. Die Konstante $\beta > 1$ ist so zu wählen, dass $\frac{\beta \log M}{n}$ ganzzahlig ist.
Gelten gewisse Einschränkungen für die Mächtigkeit der Familie $\mathcal{B}$, so kann X mit Algorithmus 3.1.1 in Polynomialzeit partiell gefärbt werden (Lemma 3.2). Dieser Vorgang lässt sich so lange mit den noch nicht gefärbten Anteilen von X wiederholen, bis dieser Rest weit genug zusammengeschrumpft ist. Abschließend lässt sich mit dem *LLL* die Existenz einer geeigneten Färbung auf diesem Rest nachweisen, die sich in weniger als M^2 Schritten durch Überprüfen aller möglichen Färbungen auffinden lässt.

Die genutzten Werkzeuge werden im Folgenden zur Übersicht verkürzt dargestellt und nicht im Einzelnen bewiesen.

3.1.1 Der Gewichtsfunktionsalgorithmus

Eingabe
- $S \subseteq X$, X eine endliche Menge.
- $B_1, \ldots, B_L$, wobei $B_i \subset X$ für alle $i \in \{1, \ldots, L\}$.
- Zu jedem B_i eine 2-Färbung $g_i : B_i \to \{\text{rot, blau}\}$.

Schritt (i) Erzeuge beliebige Permutation $\{x_1, \ldots, x_s\}$ der Elemente von S.

Schritt (ii) Färbe x_1 beliebig, setze $X_1 := \{x_1\}$.

Schritt (iii) Berechne zur zuletzt erstellten Menge X_j den Wert der Gewichtsfunktion

$$W_j^{\text{rot}} := \sum_{\substack{i \in \{1,\ldots,L\} : g|_{X_j} = g_i|_{X_j \cap B_i} \\ \text{und } g_i(x_{j+1}) = \text{rot gdw. } x_{j+1} \in B_i}} 2^{|B_i \cap (X_j \cup \{x_j+1\})|}$$

und W_j^{blau} entsprechend. Setze $X_{j+1} = X_j \cup \{x_{j+1}\}$.
Färbe x_{j+1} rot, falls $W_j^{\text{rot}} \leq W_j^{\text{blau}}$, sonst blau.

Schritt (iv) Wiederhole Schritt (iii), bis S vollständig gefärbt wurde.

Lemma 3.2. *Mit dem Gewichtsfunktionsalgorithmus wird für L nicht notwendigerweise unterschiedliche Teilmengen $B_i \subset X$ mit verbotenen Färbungen g_i eine beliebige Teilmenge $S \subseteq X$ so gefärbt, dass in allen B_i, in denen l oder mehr Elemente gefärbt wurden ($L < 2^l$), von der verbotenen Färbung abgewichen wird.*

Lemma 3.3 (Hauptlemma)**.** *Wenn $|\mathcal{B}| < 2^{(\beta \log M)/2}$, dann gilt für jede Komponente C des Abhängigkeitsgraphen von $\mathcal{H}^{(1)} = \{H_i \in \mathcal{H} : g|_{S \cap H_i} = f_i|_S\}$:*

$$|C| < \frac{\beta \log M}{n} 2^{n/12}$$

3.1.2 Becks Algorithmus

Eingabe
- Endl. Menge X
- Familie $\mathcal{H} = \{H_1, \ldots, H_M\}$ nicht notwendigerweise unterschiedlicher n-elementiger Teilmengen von X, wobei jedes H_i höchstens $2^{(n/48)+1}$ andere H_j schneidet.
- Zu jedem H_i eine 2-Färbung $f_i : H_i \to \{\text{rot, blau}\}$

Fälle
$$\begin{cases} n \geq 2^{(n/48)+1} & \text{Gehe zu Schritt (i).} \\ M < 2^n & \text{Gehe zu Schritt (ii).} \\ n < \left\lfloor 2^{(n/48)+1} \right\rfloor \text{ und } M \geq 2^n & \text{Gehe zu Schritt (iii).} \end{cases}$$

Schritt (i) Finde für alle $i \in \{1, \ldots, M\}$ Repräsentanten $x_i \in H_i$. Färbe, sodass $f(x_i) \neq f_i(x_i) \forall i$.
Gehe zu Schritt (viii).

Schritt (ii) Färbe mit Algorithmus 3.1.1 ($S = X$, $B_i = H_i$).
Gehe zu Schritt (viii).

Schritt (iii) Färbe mit Algorithmus 3.1.1 ($\{B_i\} = \mathcal{B}$). Die Menge S wird sukzessive erstellt, indem zunächst beliebige Elemente $x_1, \ldots, x_j$ gefärbt werden, bis X_j die Gleichung $max_{H_i \in \mathcal{H}} \left\{ |X_j \cap H_i| : g|_{X_j \cap H_i} = f_i|_{X_j} \right\} = \lceil n/2 \rceil$ erfüllt. x_{j+1} wird aus $X \setminus \{x \in H_i : |X_j \cap H_i| = \lceil n/2 \rceil\}$ gewählt.
Setze $\mathcal{H}^{(1)} = \{H_i \in \mathcal{H} : g|_{S \cap H_i} = f_i|_S\}$
$\mathcal{H}^{(2)} = \left\{ H_i \cap \{x \in X : x \text{ nicht gefärbt}\} : H_i \in \mathcal{H}^{(1)} \right\}$
$$\begin{cases} \text{Falls } \frac{4 \log M}{n} 2^{n/12} < 2^{\lfloor n/2 \rfloor} & \text{Gehe zu Schritt (iv).} \\ \text{Falls } \frac{4 \log M}{n} 2^{n/12} \geq 2^{\lfloor n/2 \rfloor} & \text{Gehe zu Schritt (v).} \end{cases}$$

Schritt (iv) Färbe alle Komponenten C von $\mathcal{H}^{(2)}$ vollständig mit Algorithmus 3.1.1. Die Färbung von $\mathcal{H}^{(2)}$ heiße $h = \bigcup_C g_C$.
Gehe zu Schritt (viii).

Schritt (v) Färbe $\mathcal{H}^{(2)}$ mit Algorithmus 3.1.1 entsprechend Schritt (iii). Analog zu $\mathcal{H}^{(2)}$ setze
$\mathcal{H}^{(3)} = \left\{ H_i' \cap \{x \in X : x \text{ nicht gefärbt}\} : H_i' \in \mathcal{H}^{(2)} \right\}$
$$\begin{cases} \text{Falls } \frac{15 \log \log M}{n} 2^{n/12} < 2^{n/6} & \text{Gehe zu Schritt (v).} \\ \text{Falls } \frac{15 \log \log M}{n} 2^{n/12} \geq 2^{n/6} & \text{Gehe zu Schritt (vii).} \end{cases}$$

Schritt (vi) Färbe alle Komponenten C von $\mathcal{H}^{(3)}$ vollständig mit Algorithmus 3.1.1. Die Färbung von $\mathcal{H}^{(2)}$ heiße $h = \bigcup_C g_C$.

Schritt (vii) Überprüfe alle möglichen Färbungen $\tilde{f}$ der Komponenten C von $\mathcal{H}^{(3)}$, bis
$\tilde{f}|_{H_i''} \neq f_i|_{\{x \text{ ungefärbt}\}}$ für alle $H_i'' \in \mathcal{H}^{(3)}$.

Schritt (viii) Gebe die Vereinigung $f : X \to \{\text{rot, blau}\}$ aller erstellten partiellen Färbungen aus.

Zu den Schritten des Algorithmus Die Voraussetzung für Schritt (i) ist der *Heiratssatz von Hall*. Wir finden unterschiedliche Repräsentanten $x_i \in H_i$ und färben mit f die x_i abweichend von der verbotenen Färbung f_i. Damit gilt: $f|_{H_i} \neq f_i \forall i$. Ein geeigneter Polynomialzeitalgorithmus findet sich zum Beispiel bei Schrijver in [9].
Schritt (ii) liefert nach Lemma 3.2 mit $S = X$, $L = M$, $l = n$ eine vollständige Färbung von X, die alle verbotenen f_i vermeidet.
Mit Schritt (iii) wird partiell gefärbt. Nach zufälliger Auswahl der ersten x_i wird X jeweils um die Elemente derjenigen H_i beschnitten, aus denen schon $\lceil n/2 \rceil$ Elemente ausschließlich verboten gefärbt wurden. Die Familie $\mathcal{H}^{(1)}$ enthält gerade die Mengen, deren gefärbter Anteil mit der verbotenen Färbung übereinstimmt, während in $\mathcal{H}^{(2)}$ nur noch deren ungefärbte Anteile zurückbleiben.
Mit Lemma 3.3 folgt (für $\beta = 4$, $n \geq 300$), dass alle Komponenten im Abhängigkeitsgraphen von $\mathcal{H}^{(2)}$ weniger als $\frac{4 \log M}{n} 2^{n/12}$ Ecken umfassen (wobei die Ecken Mengen $\{H_i \cap \{x \in X : x \text{ nicht gefärbt}\}\}$ entsprechen). Da jede dieser Mengen mehr als $\lfloor n/2 \rfloor$ Elemente umfasst, liefert im ersten Fall der Gewichtsfunktionsalgorithmus eine vollständige Färbung der einzelnen Komponenten und wir haben insgesamt eine geeignete Färbung gefunden (Schritt (iv)). Enthalten die Komponenten dagegen mehr als $2^{\lfloor n/2 \rfloor}$ Ecken, so kann mit Algorithmus 3.1.1 wieder nur partiell gefärbt werden (Schritt (v)). Wieder werden die verbliebenen Komponenten überprüft, im ersten Fall ist das vervollständigende Färben mit Algorithmus 3.1.1 möglich, im zweiten Fall nicht.
Bis hierher ist der ungefärbte Anteil $\mathcal{H}^{(3)}$ schon so klein, dass jede Komponente aus weniger als $(\log M)/n$ Mengen besteht, also insgesamt aus weniger als $\log M$ Elementen $x \in X$. Da jede dieser Mengen mindestens $n/6$ Elemente enthält, können wir auf jede einzelne Komponente C das LLL anwenden; n ist hinreichend groß, um mit der Zahl der benachbarten Mengen die Schranke aus 1.1.1 zu unterschreiten:

$$2^{(n/48)+1} < 2^{((n/6)-1)/e)-1}.$$

Die Zahl der möglichen Färbungen je Komponente ist kleiner als $2^{\log M} = M$, die Zahl der Komponenten ist offensichtlich kleiner M und wir können durch schlichtes Ausprobieren eine geeignete Färbung in Laufzeit $\leq M^2$.
Für die Ausgabe in Schritt (viii) werden alle nach und nach erstellten partiellen Färbungen vereinigt.

3.2 Moser und Tardos - Der konstruktive Beweis

In fast zwanzig Jahren zwischen Becks erstem Ansatz und dem Algorithmus von Moser und Tardos gab es einige Fortschritte bei den Bemühungen, die Lücke zwischen der Schranke aus dem *LLL* und der schwächeren von Beck zu verkleinern. Diese Arbeiten - unter anderem von Alon sowie Molloy und Reed - trugen dazu bei, dass mit dem in [8] vorgestellten Resultat diese Lücke vollständig geschlossen werden konnte.
Zur Beschreibung der von Moser uns Tardos verwendeten Situation benötigen wir noch einige neue Begriffe, da wir unserer bisherigen Situation einer endlichen Menge $\mathcal{A}$ von Ereignissen eine weitere Ebene hinzufügen. Wir betrachten nun eine endliche Menge $\mathcal{P}$ unabhängiger Zufallsvariablen auf einem Wahrscheinlichkeitsraum Ω, wobei jeweils Teilmengen von $\mathcal{P}$ die Ereignisse aus $\mathcal{A}$ bestimmen.

Definition 3.4. *In einem Wahrscheinlichkeitsraum Ω und für eine endliche Menge $\mathcal{P}$ von unabhängigen Zufallsvariablen, bezeichne $vbl(A)$ die Menge der bestimmenden Zufallsvariablen eines Ereignisses A, d.h. die kleinste Menge $M \subset \mathcal{P}$, sodass $A \in \sigma(M)$*

Wir nehmen an, dass uns $vbl(A)$ für jedes A zur Verwendung in unseren Algorithmen bekannt ist.

Wir betrachten außerdem den Abhängigkeitsgraphen G einer Menge von Ereignissen.

Definition 3.5. *Der Abhängigkeitsgraph $G = G_{\mathcal{A}}$ ist definiert auf der Eckenmenge $\mathcal{A}$. Eine Kante zwischen den Ecken $A \in \mathcal{A}$ und $B \in \mathcal{A}$ existiert genau dann, wenn $A \neq B$, $vbl(A) \cap vbl(B) \neq \emptyset$.*

In diesem Graphen bezeichne nun $\Gamma(A)$ die Nachbarschaft von A in G. Offensichtlich erfüllt wegen der Unabhängigkeit der Zufallsvariablen in $\mathcal{P}$ auch dieses $\Gamma(A)$ die im allgemeinen wie im symmetrischen *LLL* geforderte Eigenschaft, dass A unabhängig von $(\mathcal{A} \setminus \Gamma(A))$ sein soll.
Wir können nun den ersten Algorithmus von Moser und Tardos einführen. Dieser arbeitet mit sogenannten *Resamplings*. Startend mit einer zufälligen Realisierung der Zufallsvariablen in $\mathcal{P}$, werden immer wieder neue Realisierungen der bestimmenden Zufallsvariablen eintretender Ereignisse gezogen (*Das Ereignis wird „resampled“.*), bis eine Realisierung gefunden ist, die kein Ereignis aus $\mathcal{A}$ mehr eintreten lässt. Die erwartete Zahl der *Resamplings* eines Ereignisses $A \in \mathcal{A}$ stellt in der Analyse auch das Maß für die Effizienz des Algorithmus dar.

3.2.1 Der sequentielle Moser-Tardos-Algorithmus

Schritt (i)	Ziehe zu jeder Zufallsvariablen $P \in \mathcal{P}$ einen Wert v_P entsprechend ihrer Verteilung.
Schritt (ii)	Falls für diese Realisierung von $\mathcal{P}$ Ereignisse $A \in \mathcal{A}$ eintreten, wähle ein beliebiges eintretendes $\hat{A}$ und ziehe (unabhängig und entsprechend ihrer Verteilung) neue Werte für alle $P \in vbl(\hat{A})$.
Schritt (iii)	Wiederhole Schritt (ii), bis kein $A \in \mathcal{A}$ mehr eintritt.
Schritt (iv)	Gebe die letzte Realisierung der Variablen $P \in \mathcal{P}$ aus.

Dieser Algorithmus findet unter den Bedingungen des *LLL* tatsächlich schnell die gewünschte Realisierung, die kein $\mathcal{A}$-Ereignis mehr eintreten lässt. Dass in Schritt (ii) jeweils nur ein eintretendes Ereignis einem *Resampling* unterzogen wird, scheint unnötig viele Schritte zu erfordern. Dazu genügt folgende Überlegung: Treten für eine gegebenene Realisierung die Ereignisse $\hat{A}$ und $\tilde{A}$ mit $vbl(\hat{A}) \cap vbl(\tilde{A}) = \emptyset$ ein und wird $\hat{A}$ durch den sequentiellen Algorithmus einem *Resampling* unterzogen, so wird keine Zufallsvariable aus $vbl(\tilde{A})$ neu gezogen und $\tilde{A}$ tritt weiterhin ein. Um den Algorithmus zu beschleunigen, können die *Resamplings* parallelisiert werden:

3.2.2 Der parallele Moser-Tardos-Algorithmus

Schritt (i)	Ziehe zu jeder Zufallsvariablen $P \in \mathcal{P}$ einen Wert v_P entsprechend ihrer Verteilung.
Schritt (ii)	Falls für diese Realisierung von $\mathcal{P}$ Ereignisse $A \in \mathcal{A}$ eintreten, wähle eine maximale unabhängige Menge M im Teilgraph $G_{\hat{A}} \subset G$ der eintretenden Ereignisse und ziehe (unabhängig und entsprechend ihrer Verteilung) neue Werte für alle $P \in \bigcup_{\hat{A} \in S} vbl(\hat{A})$.
Schritt (iii)	Wiederhole Schritt (ii), bis kein $A \in \mathcal{A}$ mehr eintritt.
Schritt (iv)	Gebe die letzte Realisierung der Variablen $P \in \mathcal{P}$ aus.

Eine Teilmenge M der Eckenmenge E heißt *maximale unabhängige Menge*, wenn keine Kanten zwischen Ecken $e_i, e_j \in M, i \neq j$ existieren und keine Ecke mehr hinzugefügt werden kann, ohne dass M diese Eigenschaft verliert. Man beachte: Damit ist eine solche Menge M nicht automatisch die *größte* unabhängige Menge.

Der parallele Algorithmus ist ein Spezialfall des sequentiellen. Können wir eine obere Schranke für die erwartete Anzahl der *Resamplings* eines Ereignisses im sequentiellen Fall angeben, so gilt sie offensichtlich auch für den

parallelen Fall.

Satz 3.6 ([8] 1.2)**.** *Sei $\mathcal{P}$ eine endliche Menge unabhängiger Zufallsvariablen in einem Wahrscheinlichkeitsraum. Sei $\mathcal{A}$ eine endliche Menge von Ereignissen, bestimmt durch diese Zufallsvariablen. Existiert eine Zuordnung $x : \mathcal{A} \to (0,1)$ $(x_A := x(A))$, sodass für alle $A \in \mathcal{A}$ gilt:*

$$\mathbb{P}(A) \leq x_A \prod_{B \in \Gamma(A)} (1 - x_B),$$

dann existiert eine Realisierung der Variablen in $\mathcal{P}$, sodass kein Ereignis aus $\mathcal{A}$ eintritt.
Die Algorithmen 3.2.1 und 3.2.2 terminieren fast sicher. Sie unterziehen ein Ereignis $A \in \mathcal{A}$ dabei in Erwartung höchstens $x_A/(1 - x_A)$ Resamplings*, bevor sie eine geeignete Realisierung finden. Die erwartete Gesamtzahl von* Resamplings *ergibt sich damit als*

$$\sum_{A \in \mathcal{A}} \frac{x_A}{1 - x_A}\,.$$

Beweis. Um den Ablauf des Algorithmus nachvollziehen zu können, definieren wir das *Ausführungsprotokoll* C, das mit einer bis zur Anzahl der Schritte definierten Abbildung $C : \mathbb{N} \to \mathcal{A}$ die einem *Resampling* unterzogenen Ereignisse auflistet. Dieses C ist eine Zufallsvariable, die einerseits von den gezogenen Realisierungen der Zufallsvariablen in $\mathcal{P}$ und andererseits von dem Verfahren abhängt, mit dem in jedem Durchlauf von Schritt (ii) des sequentiellen Algorithmus ein beliebiges $\hat{A}$ für das *Resampling* ausgewählt wird. Auch, wenn diese Auswahl beliebig getroffen werden darf, nehmen wir an, dass dies nach einem festgelegten Verfahren geschieht (zufällig oder deterministisch).

Definition 3.7. *Ein* Zeugnisbaum *$\tau = (T, \sigma_T)$ ist ein endlicher gewurzelter Baum T mit einer Beschriftung $\sigma_T : V(T) \to \mathcal{A}$, $\sigma_T(u) =: [u]$ der Ecken des Baums mit Ereignissen, sodass die Nachkommen einer Ecke $u \in V(\tau)$ mit Ereignissen aus $\Gamma^+([u])$ beschriftet werden. Dabei sei $V(\tau) := V(T)$ und $\Gamma^+(A) := (\Gamma(A) \cup \{A\})$ bezeichne die* einschließliche Nachbarschaft *eines Ereignisses im Abhängigkeitsgraphen.*
Ein Zeugnisbaum heißt echt*, wenn verschiedene Nachkommen einer Ecke mit verschiedenen Ereignissen beschriftet werden.*

Für ein gegebenes Ausführungsprotokoll C, gehöre zu *Resampling* t ein Zeugnisbaum $\tau_C(t)$, der Zeugnis davon geben kann, dass unser *Resampling* tatsächlich notwendig war. Der Aufbau eines Zeugnisbaums geschieht wie folgt:

Sei $\tau_C^{(t)}(t)$ eine einzelne Ecke mit Beschriftung $C(t)$ (die Wurzel wird mit dem zuletzt einem *Resampling* unterzogenen Ereignis beschriftet). Nun gehen wir das Protokoll von rückwärts durch und unterscheiden für $i = t-1, t-2, \ldots, 1$ zwei Fälle:

(i) $\exists\, v \in \tau_C^{(i+1)}(t)$ mit $C(i) \in \Gamma^+([v])$. Wir wählen unter allen solchen v die Ecke mit dem größten Abstand von der Wurzel und hängen den Nachkommen u an diese Ecke (haben mehrere Ecken den maximalen Abstand von der Wurzel, so wählen wir unter diesen eine beliebige aus); u erhält die Beschriftung $C(i)$. Der dadurch entstandene Baum ist $\tau_C^{(i)}(t)$.

(ii) $\nexists\, v \in \tau_C^{(i+1)}(t)$ mit $C(i) \in \Gamma^+([v])$. Wir definieren $\tau_C^{(i)}(t) := \tau_C^{(i+1)}(t)$.

Nach dem vollständigen Durchlaufen dieses Erstellungsprozesses sei $\tau_C(t) := \tau_C^{(1)}(t)$.
Zur Verdeutlichung der Abläufe im Erstellungsprozess der Zeugnisbäume betrachten wir folgendes Beispiel:

Beispiel 3.1. *In Tabelle 1 ist die Belegung der fünf Klauseln einer Eingabe von* 7-SAT *mit den Boolschen Variablen b_i dargestellt (dabei steht „1“ für das Literal b_i, „0“ für $\bar{b_i}$). Entsprechend der in Behauptung 2.5 gezeigten Schranke ist diese Eingabe von* 7-SAT *erfüllbar, da jede Variable in weniger als $(2^{7-2})/7 \approx 4,57$ Klauseln auftritt.*

Klausel	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}
1	0	0	1					0	0	0	1			
2		0	1	1	0					0	1	0		
3				1	1	1	0					1	1	1
4	1					1	1	1	0				1	1
5								0	0	0	1	1	0	1

Tabelle 1: Belegung der Klauseln mit Literalen

Beschreibt A_i das Ereignis, dass die Klausel i verletzt ist (den Wert falsch *liefert), so zeigt Abbildung 1 den Abhängigkeitsgraph G dieser Ereignisse.*
Für dieses Beispiel wurde in Tabelle 2 ein möglicher Ablauf der Resamplings *dargestellt. Die Zufallsvariablen B_i liefern die Werte 0 und 1. Ist $B_i = 1$, wird die Boolsche Variable b_i mit dem Wert* wahr *belegt, für $B_i = 0$ mit* falsch*. In Schritt 1 werden alle Variablen neu gezogen. Durch diese Belegung ist das Ereignis A_5 eingetreten und dieses Ereignis wird einem* Resampling *unterzogen ($C(1) = A_5$). In der Tabelle werden in die nächste Zeile nur die Werte der neu gezogenen Variablen eingetragen. Das wird fortgesetzt, bis in*

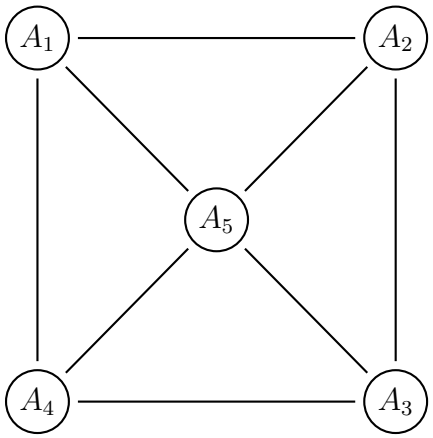

Abbildung 1: Der Abhängigkeitsgraph G

Schritt 8 kein Ereignis mehr eintritt. Die letzte Zeile stellt die Ausgabe des Algorithmus dar.

i	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	verletzt	$C(i)$
1	0	1	0	0	1	0	0	1	1	1	0	0	1	0	A_5	A_5
2								0	1	1	0	1	0	0	A_2, A_4	A_4
3	1					0	1	1	1				0	0	A_1, A_2	A_2
4		1	0	0	0					1	0	0			A_1, A_3	A_3
5				0	1	0	0					0	0	0	A_1, A_5	A_1
6	0	1	0					0	1	1	0				A_4	A_4
7	1					1	0	1	1				1	0	A_1, A_5	A_5
8								1	0	0	0	1	0	1	/	/
	1	1	0	0	1	1	0	1	0	0	0	1	0	1		

Tabelle 2: Beispielhafter Ablauf der *Resamplings*

Zu dem so erstellten Protokoll C können nun die Zeugnisbäume $\tau_C(1)$ bis $\tau_C(7)$ erstellt werden. Dieser Vorgang wird auf Seite 25 dargestellt. Dabei können wir beobachten, wie teilweise Zeugnisbäume $\tau_C(i)$ in späteren Bäumen $\tau_C(j)$ $(i < j)$ wieder auftauchen. Unterschiede können entstehen, wenn Beschriftungen ausgelassen werden, weil sich keine benachbarte Beschriftung zum Anhängen der Ecke findet oder wenn durch zufällige Auswahl bei gleichtiefen Ecken ein anderer Punkt zum Anhängen gewählt wird.

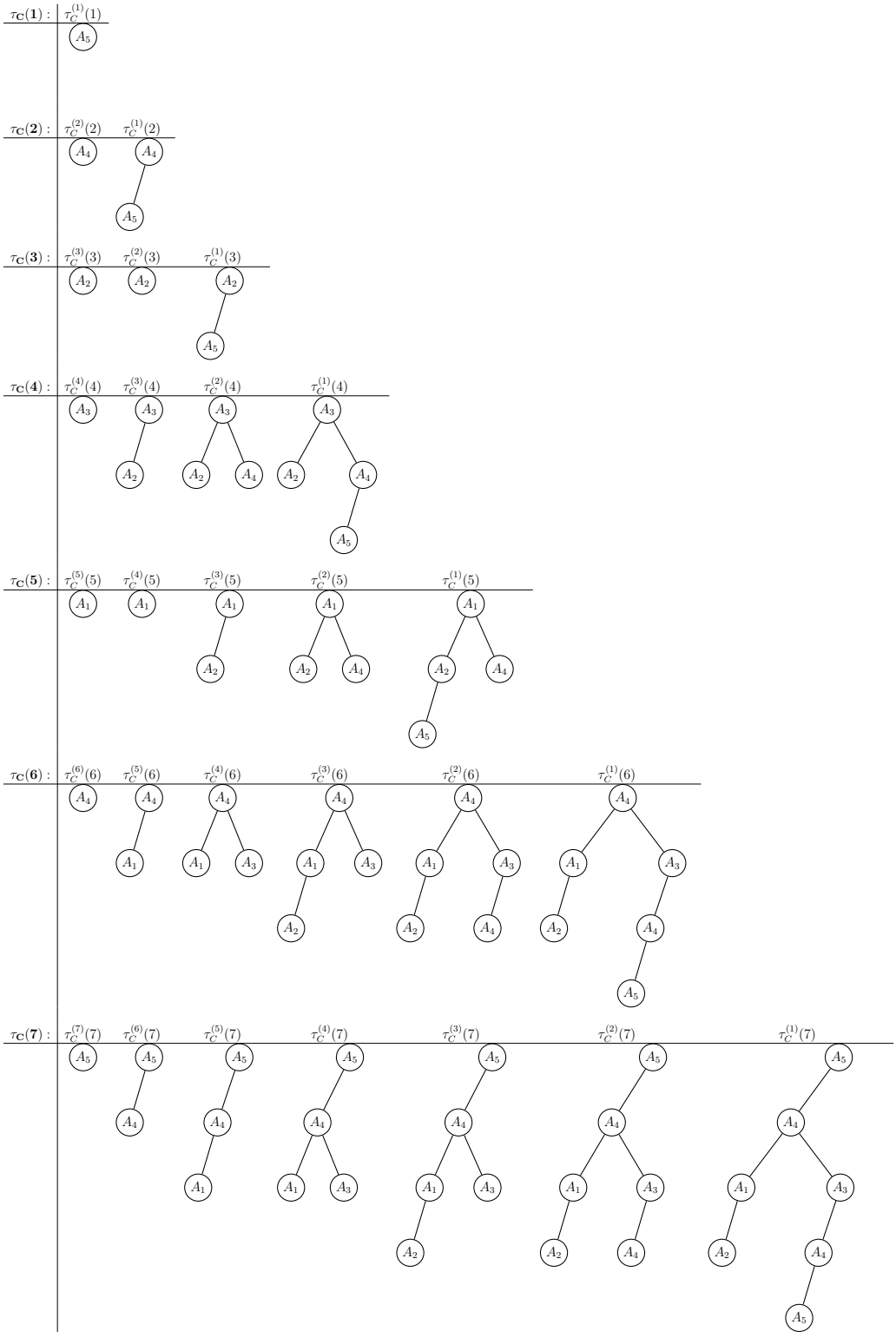

Abbildung 2: Das Wachstum der Zeugnisbäume

Lemma 3.8. *Sei τ ein Zeugnisbaum und sei C das Ausführungsprotokoll des Algorithmus. Dann gilt:*

(i) Wenn τ in C vorkommt ($\exists t \in \mathbb{N}: \tau_C(t) = \tau$), dann ist τ echt.

(ii) $\mathbb{P}(\tau \text{ kommt in } C \text{ vor}) \leq \prod_{v \in V(\tau)} \mathbb{P}([v])$.

Beweis. Die *Tiefe* einer Ecke v im Baum (der Abstand von der Wurzel) werde mit $d(v)$ bezeichnet. Mit $q(v)$ bezeichnen wir den Schritt des Erstellungsprozesses, in dem v angehängt wurde. Nach der Konstruktionsweise der Zeugnisbäume ist $q(v)$ also das größte q mit $v \in \tau_C^{(q)}(t)$.
Angenommen, τ kommt in C vor. Wir betrachten zwei Ecken $u, v \in V(\tau)$ mit $vbl([u]) \cap vbl([v]) \neq \emptyset$ und $q(u) < q(v)$. Offensichtlich ist dann $d(u) > d(v)$, denn u wird an $\tau_C^{(q(u)+1)}(t)$ angehängt, während v bereits Teil dieses Baums ist. Entsprechend wird u entweder an v oder an eine Ecke mit größerer Tiefe angehängt. Diese Erkenntnis lässt sich erweitern: Wir wissen nun, dass $[u], [v]$ nicht von gemeinsamen Variablen abhängen können, falls $d(u) = d(v)$. Damit ist τ insbesondere echt und (i) ist gezeigt.

Wir führen eine sogenannte τ-Prüfung durch. Dabei durchlaufen wir alle $v \in V(\tau)$ in einer Reihenfolge mit abnehmender Tiefe (und durchlaufen τ somit von unten nach oben). Bei Ecke v angekommen, ziehen wir eine Realisierung von $vbl([v])$ und prüfen, ob $[v]$ mit dieser Realisierung eintritt. Die τ-Prüfung sei *bestanden*, wenn alle $[v]$ mit $v \in V(\tau)$ in ihren Prüfschritten eingetreten sind. Das ist offensichtlich mit Wahrscheinlichkeit $\prod_{v \in V(\tau)} \mathbb{P}([v])$ der Fall. Es genügt also, folgendes zu zeigen:

$$\{\tau \text{ kommt in } C \text{ vor.}\} \subseteq \left\{ \begin{array}{l} \text{Die } \tau\text{-Prüfung über derselben} \\ \text{Zufallsquelle wird bestanden.} \end{array} \right\}.$$

Das scheint offensichtlich falsch zu sein: Schließlich hängen sowohl C als auch das Ergebnis der τ-Prüfung von diversen Neuziehungen der Zufallsvariablen in $\mathcal{P}$ ab. Es lohnt sich aber, die Durchführung der *Resamplings* in den Algorithmen genauer zu betrachten und dabei die Verwendung der *Zufallsquelle* zu definieren:
Der Algorithmus protokolliere zusätzlich zu den in C hinterlegten Ereignissen auch die gezogenen Realisierungen aller Zufallsvariablen $P_i \in \mathcal{P}$ und lege für jedes P_i eine Liste $\left(P_i^{(0)}, P_i^{(1)}, P_i^{(2)}, \ldots\right)$ an. Die τ-Prüfung über derselben Zufallsquelle durchzuführen heißt nun, dass in jedem Prüfschritt, in dem eine Variable P_i neu gezogen werden muss, jeweils der nächste noch nicht für die Prüfung verwendete Listeneintrag $P_i^{(j)}$ als neuer Wert für P_i benutzt wird.

Wir nehmen auch hier an, dass τ in C vorkommt und wollen zeigen, dass

die τ-Prüfung bestanden wird. Dazu muss bei jeder Überprüfung einer Ecke v die Neuziehung der Variablen in $vbl([v])$ eine Realisierung liefern, die $[v]$ eintreten lässt.
Wir betrachten eine beliebige feste Ecke v. Für $P_i \in vbl([v])$ sei

$$S(P_i) := \{w \in V(\tau) : P_i \in vbl([w]) \text{ und } d(w) > d(v)\}.$$

Ist die τ-Prüfung bei v angekommen, erhält P_i den Wert $P_i^{(|S(P_i)|)}$, denn P_i wurde bislang genau bei Überprüfung der Ecken aus $S(P_i)$ neu gezogen.
Bei Betrachtung des Protokolls C stellen wir fest, dass der Algorithmus in Schritt $q(v)$ gerade $[v]$ einem *Resampling* unterzogen hat. Das heißt, $[v]$ muss vorher eingetreten sein. Jede Variable $P_i \in vbl([v])$ wurde beim Start des Algorithmus (hier erhält jedes $P \in \mathcal{P}$ seinen ersten Wert $P^{(0)}$) und danach bei allen Schritten $q(w)$ mit $w \in S(P_i)$ und $q(w) < q(v)$ neu gezogen. Vor dem *Resampling* in Schritt $q(v)$ hat sie also den Wert $P_i^{(|S(P_i)|)}$. Das ist aber gerade diejenige Realisierung der Variablen in $vbl([v])$, die die τ-Prüfung durch die Neuziehung bei Überprüfung von v erhält. So tritt $[v]$ ein und die τ-Prüfung wird bestanden. Damit ist (ii) bewiesen. □

Die Anzahl der *Resamplings*, denen ein Ereignis $A \in \mathcal{A}$ durch einen Algorithmus unterzogen wird, sei N_A. Es lässt sich leicht feststellen, dass für $\mathcal{T}_A := \{\tau : \tau \text{ echter Zeugnisbaum, } [\text{Wurzel}] = A\}$ zugleich $N_A = |\mathcal{T}_A|$ gilt. Wir nummerieren alle $t \in \{t : C(t) = A\}$ als t_i durch. Dann ist t_i der Zeitschritt, an dem A zum i-ten Mal im Protokoll vorkommt. Der zugehörige Zeugnisbaum $\tau_C(t_i)$ enthält dann genau i Ecken mit Beschriftung A. Für $i \neq j$ ist deswegen in jedem Fall $\tau_C(t_i) \neq \tau_C(t_j)$.
Unser Ziel, die erwartete Anzahl der *Resamplings* $\mathbb{E}[N_A]$ zu beschränken, können wir also erreichen, indem wir die in Lemma 3.8 gefundene Schranke über alle $\tau \in \mathcal{T}_A$ aufsummieren. Es gilt

$$\begin{aligned}
\mathbb{E}[N_A] &= \sum_{\tau \in \mathcal{T}_A} \mathbb{E}\left[\mathbf{1}_{\{\tau \text{ kommt in } C \text{ vor}\}}\right] \\
&= \sum_{\tau \in \mathcal{T}_A} \mathbb{P}\left(\exists\, t \in \mathbb{N} : \tau_C(t) = \tau\right) \\
&\overset{\text{Lemma 3.8}}{\leq} \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \mathbb{P}([v]) \\
&\overset{\text{Ann. Satz 3.6}}{\leq} \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x([v]) \left(\prod_{B \in \Gamma([v])} (1 - x_B) \right).
\end{aligned}$$

Um diese Abschätzung im gewünschten Maße präzisieren zu können (nämlich

auf $\mathbb{E}[N_A] \leq \frac{x_A}{1-x_A}$), konstruieren wir einen *Galton-Watson-Prozess* zum Erzeugen echter Zeugnisbäume mit Wurzelbeschriftung A:

(i) Erzeuge eine Ecke v mit Beschriftung A (die Wurzel).

(ii) Betrachte alle im vorhergegangenen Schritt erzeugten Ecken. Eine Ecke v erhält zu jedem $B \in \Gamma^+([v])$ einen Nachkommen mit Beschriftung B gerade mit Wahrscheinlichkeit x_B und keinen solchen Nachkommen mit Wahrscheinlichkeit $(1 - x_B)$.

(iii) Wiederhole (ii), bis der Prozess ausstirbt.

Ob in (ii) ein bestimmter Nachkomme dem Baum hinzugefügt wird, ist also unabhängig von den Entscheidungen über die anderen möglichen Nachkommen in dieser Generation. Da Zeugnisbäume endlich sind, liefert der Prozess nicht immer einen echten Zeugnisbaum - es besteht schließlich die Möglichkeit, dass der Prozess niemals ausstirbt.

Lemma 3.9. *Sei $\tau \in \mathcal{T}_A$. Dann erzeugt der Galton-Watson-Prozess den Zeugnisbaum τ gerade mit Wahrscheinlichkeit p_τ und*

$$p_\tau = \frac{1 - x_A}{x_A} \prod_{v \in V(\tau)} \left(x([v]) \prod_{B \in \Gamma([v])} (1 - x_B) \right).$$

Beweis. Zu einer Ecke $v \in V(\tau)$ sei $G_v \subseteq \Gamma^+([v])$ die Menge der Ereignisse, mit denen die Nachkommen von v beschriftet wurden. Dann kann p_τ offensichtlich als

$$p_\tau = \frac{1}{x_A} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in (\Gamma^+([v]) \setminus G_v)} (1 - x([u])) \right)$$

geschrieben werden. Mit dem Faktor $1/x_A$ korrigieren wir, dass die als gegeben vorausgesetzte Wurzel im Produkt mit auftaucht. Fügen wir für jedes $v \in V(\tau)$ noch eine $1 = \frac{1-x([v])}{1-x([v])}$ ein (und korrigieren wieder für die Wurzel A), erhalten wir

$$p_\tau = \frac{1 - x_A}{x_A} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x([u])) \right).$$

Hier lässt sich aus dem inneren Produkt noch der Faktor für $[v]$ herauskürzen:

$$p_\tau = \frac{1 - x_A}{x_A} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in \Gamma([v])} (1 - x([u])) \right)$$

($\Gamma^+([v])$ wird zu $\Gamma([v])$ und das Lemma ist bewiesen). □

Hiermit lässt sich unsere Abschätzung für $\mathbb{E}[N_A]$ und damit auch der Beweis von Satz 3.6 vervollständigen (die letzte Abschätzung beruht auf der oben bereits gemachten Feststellung, dass der Prozess nicht immer ausstirbt und daher nicht immer einen Zeugnisbaum $\tau \in \mathcal{T}_A$ erzeugt).

$$\begin{aligned}\mathbb{E}[N_A] &\leq \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} x([v]) \left(\prod_{B\in\Gamma([v])} (1-x_B) \right) \\ &= \frac{x_A}{1-x_A} \sum_{\tau\in\mathcal{T}_A} p_\tau \\ &\leq \frac{x_A}{1-x_A}.\end{aligned}$$

□

Mit Satz 3.6 haben wir eine Schranke für $\mathbb{E}[N_A]$ gefunden, die für den sequentiellen wie für den parallelen Moser-Tardos-Algorithmus gilt. Die Ersparnis in der Anzahl der Schritte, die durch die Parallelisierung des Algorithmus entsteht, liefert uns in Verbindung mit einer etwas stärkeren Bedingung an die Wahrscheinlichkeiten der Ereignisse eine logarithmische Schranke für die erwartete Anzahl $\mathbb{E}[S]$ der Schritte des parallelen Algorithmus.

Satz 3.10 (vgl. [8] 1.3)**.** *Sei $\mathcal{P}$ eine endliche Menge unabhängiger Zufallsvariablen in einem Wahrscheinlichkeitsraum. Sei $\mathcal{A}$ eine endliche Menge von Ereignissen, bestimmt durch diese Zufallsvariablen. Existieren ein $\varepsilon > 0$ und eine Zuordnung $x : \mathcal{A} \to (0,1)$, sodass für alle $A \in \mathcal{A}$ gilt:*

$$\mathbb{P}(A) \leq (1-\varepsilon)\, x_A \prod_{B\in\Gamma(A)} (1-x_B),$$

dann ist $\mathbb{E}[S]$, die erwartete Anzahl der Schritte, die der parallele Algorithmus benötigt, bis er eine geeignete Realisierung findet und terminiert, beschränkt durch

(i) $\mathbb{E}[S] \leq \dfrac{1}{\varepsilon}$ *für* $\displaystyle\sum_{A\in\mathcal{A}} \frac{x_A}{1-x_A} \leq \frac{1}{1-\varepsilon}$,

(ii) $\mathbb{E}[S] \leq \dfrac{1}{\varepsilon}\left(1 + \log\left(\sum_{A\in\mathcal{A}} \dfrac{x_A}{1-x_A}\right)\right)$ *für* $\displaystyle\sum_{A\in\mathcal{A}} \frac{x_A}{1-x_A} > \frac{1}{1-\varepsilon}$.

Beweis. Das im Beweis von Satz 3.6 eingeführte Ausführungsprotokoll C können wir auch zur Beschreibung des parallelen Algorithmus verwenden. Nach jedem Schritt j des parallelen Algorithmus nehmen wir alle in diesem Schritt einem *Resampling* unterzogenen Ereignisse A und fügen diese

in beliebiger Reihenfolge dem Protokoll C hinzu. Die Menge der Indizes in diesem Teilstück des Protokolls heiße $I_j \subset \mathbb{N}$. Die *Höhe* eines Baums sei $h(\tau) = \max_{v \in V(\tau)} d(v)$.

Lemma 3.11. *Ist $t \in I_j$, dann ist $h(\tau_C(t)) = j - 1$.*

Beweis. Sei $t_k = \min I_k$ und sei $\tau_k = \tau_C^{(t_k)}(t)$ für $k \leq j$.
Für $k = j$ ist $\tau_j = \tau_C^{(t_j)}(t) = \tau_C^{(t)}(t)$. Da t_j und t beide in I_j liegen und alle im selben Schritt des Algorithmus einem *Resampling* unterzogenen Ereignisse unabhängig sind, werden dem Baum bis $\tau_C^{(t_j)}(t)$ keine Ecken außer der Wurzel hinzugefügt, $h(\tau_j) = 0$.
Für $k < j$ erhalten wir τ_k, indem entsprechend den in I_k gemachten *Resamplings* Ecken an τ_{k+1} angehängt werden. Dass $h(\tau_k) = h(\tau_{k+1}) + 1$ ist, zeigen wir in zwei Schritten. Zunächst betrachten wir eine Ecke $v \in \tau_{k+1}$ maximaler Tiefe. Diese Ecke steht für ein *Resampling* von $[v]$ irgendwann nach I_k. Angenommen, es existiert in τ_k keine Ecke u, sodass $d(u) > d(v)$. Dann wurde zwischen Schritt k des parallelen Algorithmus und dem *Resampling* von $[v]$ kein $w \in \Gamma^+([v])$ einem *Resampling* unterzogen und $[v]$ muss schon bei Durchführung von Schritt k eingetreten sein, was bedeutet, dass wir in diesem Schritt keine maximale unabhängige Menge für das *Resampling* ausgesucht hatten, was den Voraussetzungen des Algorithmus widerspricht. Demnach gilt $h(\tau_k) > h(\tau_{k+1})$. Da alle innerhalb von I_k angehängten Ecken mit voneinander unabhängigen Ereignissen beschriftet werden, kann die Höhe maximal um 1 steigen, $h(\tau_k) \leq 1 + h(\tau_{k+1})$. Zusammen ergibt sich $h(\tau_k) = h(\tau_{k+1}) + 1$. Es ist also $h(\tau_1 = \tau_C(t)) = \underbrace{0}_{k=j} + \underbrace{(j-1)}_{k \in [1, j-1]} \cdot 1$.

□

Sei S die Anzahl der Schritte, die der parallele Algorithmus benötigt und sei $Q(k) := \mathbb{P}(S \geq k)$ die Wahrscheinlichkeit, dass er mindestens k Schritte macht. Benötigt der parallele Algorithmus k oder mehr Schritte, so folgt aus Lemma 3.11, dass ein Zeugnisbaum τ der Höhe $h(\tau) = k - 1$ im Protokoll C vorkommen muss. Ein solcher Baum enthält offensichtlich mindestens k Ecken.
Sei $\mathcal{T}_A(k) := \{\tau \in \mathcal{T}_A : |\tau| > k\}$, also die Menge aller echten Zeugnisbäume mit Wurzel A und mehr als k Knoten.

$$
\begin{aligned}
Q(k) &\leq \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \mathbb{P}\left(\exists t \in \mathbb{N} : \tau_C(t) = \tau\right) \\
&\leq \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} \mathbb{P}([v]) \\
&\overset{\text{(Ann. Satz 3.10)}}{\leq} (1-\varepsilon)^k \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} x([v]) \left(\prod_{B\in\Gamma([v])} (1-x_B) \right) \\
&= (1-\varepsilon)^k \sum_{A\in\mathcal{A}} \frac{x_A}{1-x_A} \sum_{\tau\in\mathcal{T}_A(k)} p_\tau \\
&\leq (1-\varepsilon)^k \sum_{A\in\mathcal{A}} \frac{x_A}{1-x_A}.
\end{aligned}
$$

Aus $Q(k)$ können wir den Erwartungswert für die Anzahl der Schritte des Algorithmus berechnen.

$$\mathbb{E}\left[S\right] = \sum_{k\in\mathbb{N}} Q(k).$$

Zur Vereinfachung der Notation setzen wir im Beweis

$$\sigma := \sum_{A\in\mathcal{A}} \frac{x_A}{1-x_A}.$$

Mit der obigen Abschätzung gilt im (im Wesentlichen trivialen) Fall (i):

$$
\begin{aligned}
\mathbb{E}\left[S\right] &\leq \sum_{k\in\mathbb{N}} (1-\varepsilon)^k \sigma \\
&\leq \sum_{k\in\mathbb{N}} (1-\varepsilon)^k \frac{1}{1-\varepsilon} \\
&= \frac{1}{1-\varepsilon} \sum_{k\in\mathbb{N}} (1-\varepsilon)^k \\
&\leq \frac{1}{\varepsilon}.
\end{aligned}
$$

Zum Beweis von Fall (ii) setzen wir zunächst

$$K^* := \left\lceil \frac{\log \sigma}{\log {}^1\!/_{(1-\varepsilon)}} \right\rceil.$$

Dann gilt für alle $k \geq K^*$: $(1-\varepsilon)^k \sigma \leq 1$ und K^* ist die kleinste natürliche Zahl mit dieser Eigenschaft.
Außerdem ist

$$\varepsilon \leq \log\left(\frac{1}{1-\epsilon}\right),\ (0 < \varepsilon < 1)$$

und

$$(1-\varepsilon)^{K^*} \leq (1-\varepsilon)^{\frac{\log\sigma}{\log(1/1-\varepsilon)}} = \frac{1}{\sigma}.$$

Mit diesen Vorbetrachtungen können wir nun die Behauptung zeigen:

$$\begin{aligned}
\mathbb{E}[S] &= \sum_{k\in\mathbb{N}} Q(k) \\
&\leq \sum_{k\in\mathbb{N}} \sigma(1-\varepsilon)^k \\
&\leq \sum_{k=1}^{K^*-1} 1 + \sum_{k=K^*}^{\infty} \sigma(1-\varepsilon)^k \\
&= K^* - 1 + \sigma\frac{(1-\varepsilon)^{K^*}}{\varepsilon} \\
&\leq \left(\log\left({}^{1}\!/_{1-\varepsilon}\right)\right)^{-1}\log\sigma + \frac{1}{\varepsilon} \\
&\leq \frac{1}{\varepsilon}\left(1+\log\sigma\right).
\end{aligned}$$

□

4 Nach dem Durchbruch

4.1 Die bedingte *LLL*-Verteilung

In [5] haben Haeupler, Saha und Srinivasan 2011 die Idee einer sogenannten bedingten *LLL*-Verteilung eingeführt, die wir hier betrachten wollen, weil sie uns konstruktive Lösungsansätze für weitere Anwendungen liefern wird. Mit ihrer Hilfe können auch Anwendungen bearbeitet werden, die nicht die Bedingungen des *LLL* erfüllen, bei denen es uns aber genügt, wenn schon nicht alle, dann doch so viele unerwünschte Ereignisse wie möglich zu vermeiden.

Für den Einstieg in die Betrachtung dieser Verteilung müssen wir zunächst noch einmal zum naivstmöglichen Algorithmus zurückkehren, den man sich für die Suche nach einer günstigen Belegung der Variablen im *LLL* vorstellen kann: Wir ziehen so lange zufällige Belegungen der Variablen in $\mathcal{P}$, bis eine günstige (alle Ereignisse vermeidende) Belegung gefunden ist und geben diese als Ergebnis aus. Dass dieser Algorithmus wegen der kleinen Erfolgswahrscheinlichkeit in jedem Schritt insgesamt eine exponentielle Laufzeit erfordert, hatten wir bereits betrachtet. Insofern ist der Moser-Tardos-Algorithmus effektiver und der Algorithmus unserer Wahl.
Betrachten wir die Verteilung der von den Algorithmen ausgegebenen Belegungen, so hat der naive Algorithmus allerdings einen Vorteil: Sein Ergebnis ist eine rein zufällige Stichprobe aus der auf das Vermeiden aller unerwünschten Ereignisse bedingten Verteilung D, die wir *bedingte* LLL-*Verteilung* nennen wollen. Der Algorithmus von Moser und Tardos dagegen liefert *irgendeine* günstige Belegung. Wir wissen zunächst nicht, wie die Moser-Tardos-Ausgabeverteilung (*MTA*) auf der Menge der günstigen Belegungen aussieht. Wir können nun aber zeigen, dass die *MTA* eine gute Approximation der *LLL*-Verteilung darstellt. Wir überlegen uns zunächst eine einfache obere Schranke für die Wahrscheinlichkeit eines beliebigen von den Variablen in $\mathcal{P}$ bestimmten Ereignisses C in der *LLL*-Verteilung.
Damit die Verteilung dabei wohldefiniert ist, müssen die Bedingungen des *LLL* erfüllt sein.

Satz 4.1. *Existiert eine Zuordnung $x : \mathcal{A} \to (0, 1)$, sodass*

$$\mathbb{P}(A) \leq x_A \prod_{B \in \Gamma(A)} (1 - x_B) \quad \forall A \in \mathcal{A} \tag{2}$$

und mit dem LLL *also eine Belegung der $P \in \mathcal{P}$, sodass kein $A \in \mathcal{A}$ eintritt, dann gilt:*
Für jedes Ereignis C, das von den Variablen in $\mathcal{P}$ bestimmt wird, gilt für die

Wahrscheinlichkeit $\mathbb{P}_D(C) := \mathbb{P}\left(C \mid \bigcap_{A\in\mathcal{A}} A^c\right)$*:*

$$\mathbb{P}_D(C) \leq \mathbb{P}(C) \prod_{B\in\Gamma(C)} (1-x_B)^{-1},$$

wobei wir die bisherige Definition von $\Gamma(C)$ *auf* $C \notin \mathcal{A}$ *ausweiten. Für eine beliebige Menge* C *sei* $\Gamma(C)$ *eine Teilmenge von* $\mathcal{A}$*, sodass* C *unabhängig von* $\mathcal{A} \setminus (\Gamma(C) \cup \{C\})$ *ist.*

Beweis. Wir können die Induktion aus dem Beweis des allgemeinen *LLL* auf Seite 9 nutzen. Wir erhalten wie schon dort für die bedingte Wahrscheinlichkeit

$$\mathbb{P}(C \mid \bigcap_{A\in\mathcal{A}} A^c) = \frac{\mathbb{P}\left(C \cap \left(\bigcap_{B\in\Gamma(C)} B^c\right) \middle| \bigcap_{B\in(\mathcal{A}\setminus\Gamma(C))} B^c\right)}{\mathbb{P}\left(\bigcap_{B\in\Gamma(C)} B^c \middle| \bigcap_{B\in(\mathcal{A}\setminus\Gamma(C))} B^c\right)}$$

und können den Zähler mit $\mathbb{P}(C)$ nach oben, den Nenner mit $\prod_{B\in\Gamma(C)}(1-x_B)$ nach unten abschätzen und erhalten die behauptete Ungleichung. □

Der Satz stellt den Zusammenhang zwischen der (auf das Vermeiden aller Ereignisse in $\mathcal{A}$) bedingten Wahrscheinlichkeit für $\mathbb{P}_D(C)$ und der einfachen Wahrscheinlichkeit $\mathbb{P}(C)$ her. Umso weniger Abhängigkeiten zwischen C und Ereignissen in $\mathcal{A}$ bestehen, umso weniger wird auch die bedingte Wahrscheinlichkeit von der ursprünglichen abweichen.
Dass diese Schranke auch von der *MTA* eingehalten wird, zeigen wir im nächsten Satz:

Satz 4.2. *Sei* C *ein beliebiges Ereignis, bestimmt durch die Variablen in* $\mathcal{P}$*. Angenommen, es existiert eine Zuordnung* $x : \mathcal{A} \to (0,1)$*, sodass Gleichung (2) gilt. Dann ist die Wahrscheinlichkeit, dass* C *mindestens einmal während der Anwendung des Moser-Tardos-Algorithmus auf die Ereignisse in* $\mathcal{A}$ *eingetreten war, höchstens*

$$\mathbb{P}(C) \prod_{B\in\Gamma(C)} (1-x_B)^{-1}.$$

Diese Schranke gilt insbesondere auch für die Wahrscheinlichkeit, dass C *in der Ausgabe des Moser-Tardos-Algorithmus erfüllt ist.*

Beweis. Für den Beweis verwenden wir im Wesentlichen die Betrachtungen zu dem im Beweis von Satz 3.6 beschriebenen Galton-Watson-Prozess. Da wir die Wahrscheinlichkeit dafür suchen, dass C im Verlauf des Algorithmus irgendwann erfüllt war, betrachten wir die Zeugnisbäume, die das erste *Resampling* von C bezeugen. Das heißt diejenigen echten Zeugnisbäume mit

Wurzel C, bei denen keine weitere Ecke mit C beschriftet wurde. Die Menge dieser Bäume sei $\mathcal{T}_C$.
Wie im Beweis von Lemma 3.9 finden wir die Wahrscheinlichkeit p_τ dafür, dass der Prozess gerade einen bestimmten Baum $\tau \in \mathcal{T}_C$ erstellt.

$$p_\tau = \prod_{A \in \Gamma[C]} (1 - x_A) \cdot \prod_{v \in (V(\tau) \setminus \{B\}} \left(x([v]) \cdot \prod_{B \in \Gamma([v])} (1 - x_C) \right)$$

Exakt entlang der Abschätzungen, die wir auf Seite 27 und danach für $\mathbb{E}[N_A]$ gemacht haben (und damit eben auch für die Summe der Wahrscheinlichkeiten), können wir auch jetzt verfahren und erhalten die Schranke

$$\mathbb{P}(C) \prod_{B \in \Gamma(C)} (1 - x_B)^{-1},$$

indem wir über die Summe der p_τ abschätzen und mit $\mathbb{P}(C)$ auch das Eintreten der Wurzel C berücksichtigen. □

Wir haben nun das nötige Handwerkszeug, um eine Anwendung zu betrachten, bei der nicht alle Ereignisse, aber so viele wie möglich vermieden werden sollen. Erinnern wir uns an Bemerkung 2.7 auf Seite 13: Dort hatten wir für das k-SAT-Problem festgestellt, dass jede Klausel mit höchstens $2^k/e - 1$ anderen Klauseln Variablen „teilen“ darf, um die Erfüllbarkeit der Formel mit dem *LLL* zeigen zu können. Liegt die Zahl der benachbarten Klauseln nur knapp darüber, so können wir dazu keine Aussage mehr treffen, würden aber annehmen, dass sich noch eine große Zahl von Klauseln erfüllen lassen sollte. Das führt uns zum MAX-k-SAT-Problem. Hier sollen möglichst viele Klauseln erfüllt werden, wobei die *LLL*-Verteilung uns helfen wird.

Satz 4.3. *Angenommen, F ist eine Boolsche Formel in konjunktiver Normalform, in der eine Menge C von* Kernklauseln *existiert, sodass gilt:*

(i) Jede Klausel in C hat gemeinsame Variablen mit höchstens $d \leq 2^k/e - 1$ Klauseln in C.

(ii) Jede Klausel in $\bar{C}$ hat gemeinsame Variablen mit höchstens $\gamma(2^k/e - 1)$ Klauseln in C ($\gamma \geq 0$).

*Weiter seien n die Anzahl der Variablen, m die Zahl der Klauseln in F.
Dann gilt: Es existiert ein randomisierter Algorithmus, der mit hoher Wahrscheinlichkeit in $\text{poly}(m, n)$-Zeit eine Belegung der Variablen erzeugt, die keine Klausel in C und in Erwartung einen Anteil von höchstens $2^{-k}e^\gamma$ der Klauseln in $\bar{C}$ verletzt.*

Beweis. Wir betrachten wieder eine zufällige Belegung der Boolschen Variablen. Jede Variable sei unabhängig von den anderen *wahr* oder *falsch* jeweils mit Wahrscheinlichkeit 1/2. Zur Klausel C_i sei A_i das unerwünschte Ereignis „C_i ist nicht erfüllt“. Wie wir im Beweis von Satz 2.3 gesehen haben, können wir für alle C_i identische Werte $x_{A_i} = 1/(d+1) = e/2^k$ wählen. Mit dem symmetrischen *LLL* und dem Algorithmus von Moser und Tardos finden wir eine Belegung, die alle Ereignisse A_i vermeidet.
Lassen wir den Algorithmus n^c-mal seine erwartete Zahl von *Resamplings* $\mathbb{E}[N_A]$ laufen, so erhalten wir mit der Markov-Ungleichung eine Abschätzung für die Wahrscheinlichkeit, dass er bis zu diesem Zeitpunkt noch nicht terminiert:

$$\mathbb{P}\left(N_A > n^c \,\mathbb{E}[N_A]\right) \leq \frac{1}{n^c \,\mathbb{E}[N_A]} \,\mathbb{E}[N_A] = n^{-c}.$$

Darüber hinaus kann die Wahrscheinlichkeit des Ereignisses B_i, dass eine Klausel $C_i \in \bar{C}$ noch nicht erfüllt ist, zu diesem Zeitpunkt mit Satz 4.2 abgeschätzt werden durch

$$\begin{aligned}
\mathbb{P}(B_i) &\overset{Satz 4.2}{\leq} \mathbb{P}(B_i \mid \text{Zufällige Belegung}) \cdot \left(\prod_{C \in \Gamma(B_i)} (1 - x_C)\right)^{-1} \\
&\leq 2^{-k} \left(\left(1 - \frac{e}{2^k}\right)^{\gamma\left(2^k/e - 1\right)}\right)^{-1} \\
&= 2^{-k} \left(1 - \frac{e}{2^k}\right)^{-\gamma\left(2^k/e - 1\right)},
\end{aligned}$$

wobei wir die Tatsache nutzen, dass Satz 4.2 auch für Ereignisse außerhalb der *LLL*-Ereignismenge gilt. Für $k \geq 3$ können wir dies durch Abschätzen von $(2^k/e - 1) \geq 1$ und $(1 - e/2^k) < e$ vereinfachen zu

$$\mathbb{P}(B_i) \leq 2^{-k}\, e^{\gamma},$$

für $k = 1$ und $k = 2$ lässt sich die Richtigkeit dieser Abschätzung durch Einsetzen überprüfen. Also ist der erwartete Anteil nicht erfüllter Klauseln in $\bar{C}$ höchstens $2^{-k}\, e^{\gamma}$. □

Darüber hinaus konnten die Autoren in [5] eine konstruktive Technik entwickeln, mit der solche Mengen von Kernklauseln in einer Boolschen Formel effektiv gefunden werden können.

4.2 Ein Argument mit Ausstrahlung?

Eine weitere Betrachtung des Beitrags von Moser und Tardos soll den Abschluss dieser Arbeit bilden. Dabei geht es nicht um die Aussage zum *LLL*, sondern vielmehr um die Idee hinter dem Beweis.
In seinem Beitrag zu *Moser's entropy compression argument* [10] rückte Terence Tao die Beweistechnik Mosers in den Vordergrund und postulierte, es sei zu erwarten, dass diese auf weitere Fragestellungen aus dem Bereich randomisierter Algorithmen anwendbar sein würde. Wir werden hier Taos Blickwinkel folgen und die *Entropiekompression* genauer betrachten.

Wir betrachten die Frage, ob ein Algorithmus terminiert. Es sind einige Techniken bekannt, mit deren Hilfe sich zeigen lässt, dass ein Algorithmus, der durch Iteration Schritt für Schritt bessere Versionen eines Objekts erzeugt, terminieren muss, wenn mit jedem Schritt eine streng monotone Veränderung in einer bestimmten beschränkten Größe eintritt; zum Beispiel, indem gezeigt werden kann, dass die Größe einer Teilmenge einer endlichen Menge mit jeder Iteration um 1 wächst und somit feststeht, dass dieser Prozess nicht unendlich fortgesetzt werden kann.
Mosers Beweistechnik kann als Entropiekompression bezeichnet werden. Diese lässt sich auf randomisierte Algorithmen anwenden, die als Teil ihrer Eingabe eine Menge R von Realisierungen von Zufallsvariablen erhalten. Im Fall des Moser-Tardos-Algorithmus sind dies die $(P_i^{(0)}, P_i^{(1)}, P_i^{(2)}, \ldots)$, die die Realisierungen der Variablen in $\mathcal{P}$ enthalten.
Jeder Schritt des Algorithmus nimmt das zu verbessernde Objekt A und die zufällige Zeichenkette R und erzeugt aus A und einem Teil von R eine bessere Version A' und eine verkürzte Kette R', die nur aus den noch nicht verwendeten Einträgen von R besteht. Wichtig ist dabei, durch Anlage eines Verlaufs H den Algorithmus umkehrbar zu machen (also die Möglichkeit offen zu lassen, aus A', R' und H auf die vorangegangenen A und R zu schließen). Da der gesamte Informationsgehalt aus $A + R$ verlustfrei in $A' + R' + H$ komprimiert werden soll, kann die neue Zeichenkette in Erwartung nicht kürzer als die Entropie der Zufallsvariable $A + R$ werden. Kennen wir nun also eine geeignete Schranke für die Entropie und ist der Zuwachs an Stellen pro Schritt im Verlauf H kleiner als die Zahl der pro Schritt benötigten Zufallsstellen von R, dann haben wir ein Monotonieargument von der oben beschriebenen Bauart, das uns das Terminieren des Algorithmus garantiert.
Die Untersuchung dieses Arguments in Bezug auf den Moser-Tardos-Algorithmus wollen wir wie in [10] der leichteren Verständlichkeit halber auf die Anwendung auf das k-SAT-Problem beschränken, die in dieser Arbeit schon in allen Kapiteln der verständlichen Darstellung der jeweiligen Problematik

gedient hat.

Wir betrachten den etwas vereinfachten Fall, dass jede Klausel gemeinsame Variablen mit höchstens $2^{k-C}-1$ anderen Klauseln hat ($C \geq \log_2(e)+1$ eine Konstante) und erhalten als Spezialfall des *LLL* die folgende

Behauptung 4.4. *Sei S eine Menge von Klauseln der Länge k und C eine hinreichend große Konstante. Wenn jede Klausel gemeinsame Variablen mit höchstens $2^{k-C}-1$ anderen Klauseln in S hat, dann sind alle Klauseln in S gleichzeitig erfüllbar.*

Wir beginnen wie bisher mit einer zufälligen Belegung A der Boolschen Variablen. Die Teilmenge der nicht erfüllten Klauseln in unserer Boolschen Formel heiße T. Sind alle Klauseln erfüllt ($|T| = 0$), dann terminiert der Algorithmus schon jetzt. Ist $|T| > 0$, dann können wir nach und nach die Klauseln in T durchzugehen und jeweils k neue Zufallswerte aus der Zeichenkette R verwenden, um neue Realisierungen der k Variablen zu ziehen, die in der jeweiligen Klausel vorkommen. Dass dabei jeweils bis zu $2^{k-C}-1$ andere Klauseln verletzt werden können, die bislang erfüllt waren, lässt dieses Verfahren scheinbar niemals enden. Da aber jede nicht erfüllte Klausel k Stellen an Information über A preisgibt, kommen wir mit unserer Kompression voran.
Um genauer in die Überprüfung der Kompression einzusteigen, benötigen wir den folgenden Algorithmus, der zur Reparatur einer Klausel s verwendet wird. Wir bezeichnen dabei entsprechend der Notation in Kapitel 2 mit $B(s)$ die Menge der Boolschen Variablen, die in s auftreten. Darüber hinaus definieren wir $B_A(s)$ als die Belegung der Variablen in $B(s)$ duch A.

4.2.1 Der *Fix(s)*-Algorithmus

Schritt (i)	Wenn A die Klausel s bereits erfüllt, lasse A unverändert und gebe $A' = A$ aus. Wenn A die Klausel s nicht erfüllt, nehme die ersten k Stellen aus R und ersetze mit ihnen die Werte der Variablen in s. R' sei R ohne die verwendeten Stellen.
Schritt (ii)	Finde alle verletzten Klauseln s' mit $B(s') \cap B(s) \neq \emptyset$. Wende *Fix(s')* nacheinander auf alle s' an. Gehe zu Schritt (i).

Wenn ein aufgerufener *Fix(s)* terminiert, dann erfüllt A' die Klausel s und mindestens alle Klauseln s', die vor dem Aufruf von *Fix(s)* erfüllt waren. Das heißt, *Fix(s)* verringert $|T|$. Wir überlegen uns nun, dass die gewünschte Umkehrbarkeit vorhanden ist und sich A und R aus s, A' und R' rekonstruieren

lassen. A erhalten wir, indem wir $B_{A'}(s)$ durch die verbotene Belegung von s ersetzen, $R = B_{A'}(s) + R'$.
Was bedeutet das für die Längen der Zeichenketten? Wir speichern $A'+R'+s$ statt $A + R$. A' und A sind gleich lang, R' ist k Stellen kürzer als R und für das Speichern einer beliebigen Klausel s benötigen wir etwa $\log |S|$ Stellen. Eine Ersparnis in der Zahl der Stellen erreichen wir also scheinbar nur für $\log |S| < k$.
Durch den Algorithmus *Fix(s)* lässt sich nach einigen Schritten aber Speicherplatz einsparen. Zwar muss jede der $|T|$ anfangs verletzten Klauseln mit $\log |S|$ Stellen gespeichert werden, bei den aus Schritt (ii) eines *Fix(s)* aufgerufenen Läufen von *Fix* genügt aber eine Zahl: Für jedes s nummerieren wir die höchstens 2^{k-C} benachbarten Klauseln s' durch und speichern beim Aufruf von *Fix(s')* statt s' nur dessen *Index* in dieser Liste. Während *Fix* läuft, brauchen wir nur einen Verlauf H der Form

$$s_1\ i_1|\ \ j_1|\ \ k_1||\ \ s_2\ i_2|\ \ j_2|\ \ k_2||\ \ s_3\ \ i_3|\ \ldots||$$

zu speichern. Dabei stehen i, j, k für die Indizes, die senkrechten Striche stellen ein vom Algorithmus gespeichertes *Stoppsymbol* dar, das angibt, wann ein *Fix* terminiert. Aus diesem Verlauf H lässt sich nun zu jedem Zeitpunkt rekonstruieren, welche Klausel gerade repariert wird und somit nach und nach auch der ursprüngliche Zustand $A + R$.
Der Beweis lässt sich nun durch unser Entropieargument abschließen, wenn wir annehmen, die Klauseln aus S könnten nicht alle zugleich erfüllt werden. Unter dieser Annahme kann die Reihe der *Fix*-Algorithmen nicht terminieren. Nehmen wir weiter an, die zufällige Zeichenkette R habe zu Beginn die Länge $M \cdot k$. Dadurch können M *Fix*-Läufe aufgerufen werden, bis alle Stellen verwendet wurden und die Kette R' leer ist. Die Länge des Verlaufs H dagegen ist auf höchstens $O(|S|) + M(k - C + O(1))$ Stellen angewachsen. Dieser Wert ergibt sich aus $|S|$ Stellen, um die Klauseln zu speischern, die anfangs in T sind, außerdem $O(|S|)+O(M)$ Stellen zum Speichern aller Aufrufe von Schritt (i) des *Fix*-Algorithmus und für jeden Aufruf von *Fix* aus Schritt (iii) eines übergeordneten Laufs $(k - C)$ Stellen und gegebenenfalls $O(1)$ Stellen für das Stoppsymbol.
Insgesamt hätten wir also durch den Algorithmus die Information aus $A + R$ verlustfrei in $A' + H$ komprimiert und damit $n + Mk$ zufällige Stellen (A war zu Anfang ja ebenfalls mit einer von R unabhängigen zufälligen Belegung der Boolschen Variablen gefüllt worden) auf $n + O(|S|) + M(k - C + O(1))$ Stellen komprimiert. Da aber $n + Mk$ zufällige Stellen nicht verlustfrei in eine kürzere Zeichenkette komprimiert werden können, liefert die Entropie

uns die folgende Schranke:

$$n + O(|S|) + M(k - C + O(1)) \geq n + Mk.$$

Dies erzeugt, falls C größer als eine gewisse Konstante ist, für hinreichend große M einen Widerspruch. M kann aber zu Beginn bei der Erzeugung von R frei gewählt werden. Das heißt, es können doch alle Klauseln aus S gleichzeitig erfüllt werden und Behauptung 4.4 ist mithilfe der Entropiekompression bewiesen.

Ausblick

Mit diesem Einblick in zwei Reaktionen auf die Arbeit von Moser und Tardos wurde selbstverständlich nur ein kleiner Teil des reichhaltigen Schatzes von Publikationen betrachtet, die die Untersuchung konstruktiver Ergebnisse zum *LLL* fortsetzen.
Es muss dem Leser überlassen werden, das hier geschaffene grundsätzliche Verständnis für Idee und Nutzen des *LLL* und seiner konstruktiven Aspekte zum vertiefenden Studium weiterer aktueller Arbeiten zu verwenden.

Literatur

[1] Noga Alon und Joel H. Spencer *The probabilistic method, Second Edition*, John Wiley & Sons, Inc., 2000

[2] J. Beck *An algorithmic approach to the Lovász Local Lemma*, in: *Random Structures & Algorithms*, 2(4): 343-365, 1991

[3] Paul Erdős und László Lovász *Problems and results on 3-chromatic hypergraphs and some related questions* in: Hajnal/Rado/Sos (Hrsg.) *Infinite and finite sets*, pages 609-627, 1975

[4] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, B. Reed (Hrsg.), *Probabilistic Methods for Algorithmic Discrete Mathematics*, Springer, 1998

[5] Bernhard Haeupler, Barna Saha und Aravind Srinivasan *New constructive Aspects of the Lovász Local Lemma*, in: Journal of the ACM 58, 6, Art. 28, Dezember 2011

[6] F. T. Leighton, Bruce M. Maggs, Satish B. Rao *Packet Routing and Job-Shop Scheduling in O(Congestion + dilation) Steps*, in: *Combinatorica*, 14: 167-186, 1994

[7] Michael Molloy und Bruce Reed, *Graph Colouring and the Probabilistic Method*, Springer, 2002

[8] Robin A. Moser und Gábor Tardos *A constructive proof of the general Lovász Local Lemma*, in: Journal of the ACM 57, 2, Art. 11, Januar 2010

[9] Alexander Schrijver, *Combinatorial Optimization*, Springer, 2003;
Der Algorithmus findet sich in Abschnitt 16.3 (Band A, S. 263).

[10] Terence Tao, *Moser's entropy compression argument*, Beitrag vom 5.8.2009 im Blog *What's new*, URL: terrytao.wordpress.com/2009/08/05/mosers-entropy-compression-argument/, abgerufen 28. März 2012